# conda

*Release 25.5.2.dev51*

**Anaconda, Inc.**

**Jul 03, 2025**

# CONTENTS

Welcome to conda's documentation! Conda provides package, dependency, and environment management for any language. Here, you will find everything you need to get started using conda in your own projects.

CHAPTER

ONE

# INSTALL

We recommend the following conda distribtions to install conda:

Miniconda Miniconda is an installer by Anaconda that comes preconfigured for use with the Anaconda Repository. See the notes about Anaconda's *Terms of Service*.

Windows x86_64

macOS arm64 (Apple Silicon)

macOS x86_64 (Intel)

Linux x86_64 (amd64)

Linux aarch64 (arm64)

Or with Homebrew:

```
brew install miniconda
```

Miniforge Miniforge is an installer maintained by the conda-forge community that comes preconfigured for use with the conda-forge channel.

Windows x86_64

macOS arm64 (Apple Silicon)

macOS x86_64 (Intel)

Linux x86_64 (amd64)

Linux aarch64 (arm64)

Or with Homebrew:

```
brew install miniforge
```

CHAPTER

TWO

# NEW TO CONDA?

If you are new to conda, we first recommend the following articles:

Getting started guide  Learn the basics of using conda such as creating and adding packages to environments

Managing environments  Learn more about environments and best practices for using them in your projects

**See also**

Want to get even more in-depth training on how to use conda for free? Check out Anaconda's free course on conda basics.

CHAPTER

THREE

# OTHER USEFUL RESOURCES

Command reference  Full reference for all standard commands and options

Cheatsheets  Get the latest cheatsheet for commonly used commands

Configuring conda  Learn about the various ways conda's behavior can be configured

Glossary  Important vocabulary to know when working with conda

CHAPTER

FOUR

# CONTRIBUTORS WELCOME

Conda is an open source project and always welcomes new contributions. Please read the following guides to get started developing conda and making your own contributions.

Contributing 101 Learn more about how the conda project is managed and how to contribute

Development environment Follow this guide to get your own development environment set up

## 4.1 User guide

### 4.1.1 Getting started with conda

Conda is a powerful command line tool for package and environment management that runs on Windows, macOS, and Linux.

This guide to getting started with conda goes over the basics of starting up and using conda to create environments and install packages.

**Tip**

Anaconda Navigator is a graphical desktop application that enables you to use conda without having to run commands at the command line.

See Getting started with Anaconda Navigator to learn more.

#### Before you start

To bootstrap a `conda` installation, use a minimal installer such as Miniconda or Miniforge.

Conda is also included in the Anaconda Distribution.

**Note**

Miniconda and Anaconda Distribution come preconfigured to use the Anaconda Repository and installing/using packages from that repository is governed by the Anaconda Terms of Service, which means that it *might* require a commercial fee license. There are exceptions for individuals, universities and companies with fewer than 200 employees (as of September 2024).

Please review the terms of service, Anaconda's most recent Update on Anaconda's Terms of Service for Academia and Research, and the Anaconda Terms of Service FAQ to answer your questions.

### Starting conda

Conda is available on Windows, macOS, or Linux and can be used with any terminal application (or shell).

### Windows

1. Open either the Anaconda or Miniforge Command Prompt (cmd.exe). A PowerShell prompt is also available with Anaconda Distribution or Miniconda.

### macOS

1. Open Launchpad.
2. Open the Other application folder.
3. Open the Terminal application.

### Linux

Open a terminal window.

### Creating environments

Conda allows you to create separate environments, each containing their own files, packages, and package dependencies. The contents of each environment do not interact with each other.

The most basic way to create a new environment is with the following command:

```
conda create -n <env-name>
```

To add packages while creating an environment, specify them after the environment name:

```
conda create -n myenvironment python numpy pandas
```

For more information on working with environments, see *Managing environments*.

### Listing environments

To see a list of all your environments:

```
conda info --envs
```

A list of environments appears, similar to the following:

```
conda environments:

   base           /home/username/Anaconda3
   myenvironment   * /home/username/Anaconda3/envs/myenvironment
```

> **Tip**
>
> The active environment is the one with an asterisk (*).

To change your current environment back to the default one:

```
conda activate
```

> **Tip**
>
> When the environment is deactivated, its name is no longer shown in your prompt, and the asterisk (*) returns to the default env. To verify, you can repeat the `conda info --envs` command.

### Installing packages

You can also install packages into a previously created environment. To do this, you can either activate the environment you want to modify or specify the environment name on the command line:

```
# via environment activation
conda activate myenvironment
conda install matplotlib

# via command line option
conda install --name myenvironment matplotlib
```

For more information on searching for and installing packages, see *Managing packages*.

### Specifying channels

Channels are locations (on your own computer or elsewhere on the Internet) where packages are stored. By default, conda searches for packages in its *default channels*.

If a package you want is located in another channel, such as conda-forge, you can manually specify the channel when installing the package:

```
conda install conda-forge::numpy
```

You can also override the default channels in your *.condarc* file. For a direct example, see *Channel locations (channels)* or read the entire *Using the .condarc conda configuration file*.

> **Tip**
>
> Find more packages and channels by searching Anaconda.org.

### Updating conda

To see your conda version, use the following command:

```
conda --version
```

No matter which environment you run this command in, conda displays its current version:

```
conda 25.5.2.dev51
```

> **Note**
>
> If you get an error message `command not found: conda`, close and reopen your terminal window and verify that you are logged into the same user account that you used to install conda.

To update conda to the latest version:

```
conda update conda
```

Conda compares your version to the latest available version and then displays what is available to install.

> **Tip**
>
> We recommend that you always keep conda updated to the latest version. For conda's official version support policy, see CEP 10.

### More information

- *Conda cheat sheet*
- Full documentation
- Free community support

## 4.1.2 Installing conda

To install conda, you must first pick the right installer for you. The following are the most popular installers currently available:

**Miniconda**
: Miniconda is a minimal installer provided by Anaconda. Use this installer if you want to install most packages yourself.

**Anaconda Distribution**
: Anaconda Distribution is a full featured installer that comes with a suite of packages for data science, as well as Anaconda Navigator, a GUI application for working with conda environments.

**Miniforge**
: Miniforge is an installer maintained by the conda-forge community that comes preconfigured for use with the conda-forge channel. To learn more about conda-forge, visit their website.

**Note**

Miniconda and Anaconda Distribution come preconfigured to use the Anaconda Repository and installing/using packages from that repository is governed by the Anaconda Terms of Service, which means that it *might* require a commercial fee license. There are exceptions for individuals, universities, and companies with fewer than 200 employees (as of September 2024).

Please review the terms of service, Anaconda's most recent Update on Anaconda's Terms of Service for Academia and Research, and the Anaconda Terms of Service FAQ to answer your questions.

### System requirements

- A supported operating systems: Windows, macOS, or Linux
- For Miniconda or Miniforge: 400 MB disk space
- For Anaconda: Minimum 3 GB disk space to download and install
- For Windows: Windows 8.1 or newer for Python 3.9

**Tip**

You do not need administrative or root permissions to install conda if you select a user-writable install location (e.g. `/Users/my-username/conda` or `C:\Users\my-username\conda`).

### Regular installation

Follow the instructions for your operating system:

- *Windows*
- *macOS*
- *Linux*

### Installing in silent mode

You can use *silent installation* of Miniconda, Anaconda, or Miniforge for deployment or testing or building services, such as GitHub Actions.

Follow the silent-mode instructions for your operating system:

- *Windows*
- *macOS*
- *Linux*

### Cryptographic hash verification

SHA-256 checksums are available for Miniconda and Anaconda Distribution. We do not recommend using MD5 verification as SHA-256 is more secure.

Download the installer file and, before installing, verify it as follows:

- Windows:
  - If you have PowerShell V4 or later:

    Open a PowerShell console and verify the file as follows:

    ```
    Get-FileHash filename -Algorithm SHA256
    ```

  - If you don't have PowerShell V4 or later:

    Use the free online verifier tool on the Microsoft website.

    1. Download the file and extract it.
    2. Open a Command Prompt window.
    3. Navigate to the file.
    4. Run the following command:

       ```
       Start-PsFCIV -Path C:\path\to\file.ext -HashAlgorithm SHA256 -Online
       ```

- macOS: In iTerm or a terminal window enter `shasum -a 256 filename`.
- Linux: In a terminal window enter `sha256sum filename`.

### Installing on Windows

1. Download the installer:
   - Miniconda installer for Windows
   - Anaconda Distribution installer for Windows
   - Miniforge installer for Windows
2. *Verify your installer hashes*.
3. Double-click the `.exe` file.
4. Follow the instructions on the screen.

   If you are unsure about any setting, accept the defaults. You can change them later.

   When installation is finished, from the **Start** menu, open either the Anaconda Command Prompt (cmd.exe) if using Miniconda or Anaconda Distribution, and the Miniforge Command Prompt if using Miniforge. Powershell prompts are also available.
5. Test your installation. In your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

### Installing in silent mode

> **Note**
>
> The following instructions are for Miniconda but should also work for the Anaconda Distribution or Miniforge installers.

> **Note**
>
> As of `Anaconda Distribution 2022.05` and `Miniconda 4.12.0`, the option to add Anaconda to the PATH environment variable during an **All Users** installation has been disabled. This was done to address a security exploit. You can still add Anaconda to the PATH environment variable during a **Just Me** installation.

To run the the Windows installer for Miniconda in *silent mode*, use the `/S` argument. The following optional arguments are supported:

- `/InstallationType=[JustMe|AllUsers]`---Default is `JustMe`.
- `/AddToPath=[0|1]`---Default is `0`
- `/RegisterPython=[0|1]`---Make this the system's default Python. `0` indicates Python won't be registered as the system's default. `1` indicates Python will be registered as the system's default.
- `/S`---Install in silent mode.
- `/D=<installation path>`---Destination installation path. Must be the last argument. Do not wrap in quotation marks. Required if you use `/S`.

All arguments are case-sensitive.

**Example:** The following command installs Miniconda for the current user without registering Python as the system's default:

```
start /wait "" Miniconda3-latest-Windows-x86_64.exe /InstallationType=JustMe /
↪RegisterPython=0 /S /D=%UserProfile%\Miniconda3
```

### Updating conda

1. Open Command Prompt or PowerShell from the start menu.
2. Run `conda update conda`.

### Uninstalling conda

1. In the Windows Control Panel, click Add or Remove Program.
2. Select Python X.X (Miniconda), where X.X is your version of Python.
3. Click Remove Program.

> **Note**
>
> Removing a program is different in Windows 10.

### Installing on macOS

**Caution**

If you use the `.pkg` installer for Miniconda, beware that those installers may skip the "Destination Select" page which will cause the installation to fail. If the installer skips this page, click "Change Install Location..." on the "Installation Type" page, choose a location for your install, and then click Continue.

1. Download the installer:
   - Miniconda installer for macOS.
   - Anaconda Distribution installer for macOS.
   - Miniforge installer for macOS.
2. *Verify your installer hashes*.
3. Install:
   - Miniconda or Miniforge: in your terminal window, run:

     ```
     bash <conda-installer-name>-latest-MacOSX-x86_64.sh
     ```

   - Anaconda Distribution: double-click the `.pkg` file.
4. Follow the prompts on the installer screens. If you are unsure about any setting, accept the defaults. You can change them later.
5. To make the changes take effect, close and then re-open your terminal window.
6. To verify your installation, in your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

### Installing in silent mode

**Note**

The following instructions are for Miniconda but should also work for the Anaconda Distribution or Miniforge installers.

To run the *silent installation* of Miniconda for macOS or Linux, specify the -b and -p arguments of the bash installer. The following arguments are supported:

- `-b`: Batch mode with no PATH modifications to shell scripts. Assumes that you agree to the license agreement. Does not edit shell scripts such as `.bashrc`, `.bash_profile`, `.zshrc`, etc.
- `-p`: Installation prefix/path.
- `-f`: Force installation even if prefix `-p` already exists.

**Example**

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh -O ~/
↪miniconda.sh
bash ~/miniconda.sh -b -p $HOME/miniconda
```

**Note**

In order to initialize after the installation process is done, first run `source <path to conda>/bin/activate` and then run `conda init --all`.

### Updating Anaconda or Miniconda

1. Open a terminal window.
2. Run `conda update conda`.

### Uninstalling Anaconda or Miniconda

1. Open a terminal window.
2. Remove the entire Miniconda install directory with (*this may differ depending on your installation location*)

```
rm -rf ~/miniconda
```

3. *Optional*: run `conda init --reverse --all` to undo changes to shell initialization scripts
4. *Optional*: remove the following hidden file and folders that may have been created in the home directory:
   - `.condarc` file
   - `.conda` directory
   - `.continuum` directory

   By running:

```
rm -rf ~/.condarc ~/.conda ~/.continuum
```

### Installing on Linux

1. Download the installer:
   - Miniconda installer for Linux.
   - Anaconda Distribution installer for Linux.
   - Miniforge installer for Linux.
2. *Verify your installer hashes*.
3. In your terminal window, run:

```
bash <conda-installer-name>-latest-Linux-x86_64.sh
```

   `conda-installer-name` will be one of "Miniconda3", "Anaconda", or "Miniforge3".
4. Follow the prompts on the installer screens. If you are unsure about any setting, accept the defaults. You can change them later.
5. To make the changes take effect, close and then re-open your terminal window.
6. Test your installation. In your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.

### Using with fish shell

To use conda with fish shell, run the following in your terminal:

Add conda binary to $PATH, if not yet added:

```
fish_add_path <conda-install-location>/condabin
```

Configure fish-shell:

```
conda init fish
```

### Installing in silent mode

See the instructions for *installing in silent mode on macOS*.

### Updating conda

1. Open a terminal window.
2. Run `conda update conda`.

### Uninstalling conda

1. Open a terminal window.
2. Remove the entire conda install directory with (*this may differ depending on your installation location*)

   ```
   rm -rf ~/conda
   ```

3. *Optional*: run `conda init --reverse --all` to undo changes to shell initialization scripts
4. *Optional*: remove the following hidden file and folders that may have been created in the home directory:
   - `.condarc` file
   - `.conda` directory
   - `.continuum` directory

   By running:

   ```
   rm -rf ~/.condarc ~/.conda ~/.continuum
   ```

### RPM and Debian Repositories for Miniconda

Conda is available as either a RedHat RPM or as a Debian package. The packages are the equivalent to the Miniconda installer, which only contains conda and its dependencies. You can use yum or apt to install, uninstall, and manage conda on your system. To install conda, follow the instructions for your Linux distribution.

To install the RPM on RedHat, CentOS, Fedora distributions, and other RPM-based distributions, such as openSUSE, download the GPG key and add a repository configuration file for conda.

```
# Import our GPG public key
rpm --import https://repo.anaconda.com/pkgs/misc/gpgkeys/anaconda.asc

# Add the Anaconda repository
cat <<EOF > /etc/yum.repos.d/conda.repo
[conda]
name=Conda
baseurl=https://repo.anaconda.com/pkgs/misc/rpmrepo/conda
enabled=1
gpgcheck=1
gpgkey=https://repo.anaconda.com/pkgs/misc/gpgkeys/anaconda.asc
EOF
```

Conda is ready to install on your RPM-based distribution.

```
  # Install it!
  yum install conda
  Loaded plugins: fastestmirror, ovl
  Setting up Install Process
  Loading mirror speeds from cached hostfile
   * base: repo1.dal.innoscale.net
   * extras: mirrordenver.fdcservers.net
   * updates: mirror.tzulo.com
  Resolving Dependencies
  --> Running transaction check
  ---> Package conda.x86_64 0:4.5.11-0 will be installed
  --> Finished Dependency Resolution

  Dependencies Resolved


================================================================================
  Package         Arch        Version             Repository            Size

================================================================================
  Installing:
  conda           x86_64      4.5.11-0            conda                 73 M

  Transaction Summary

================================================================================
  Install      1 Package(s)

  Total download size: 73 M
  Installed size: 210 M
  Is this ok [y/N]:
```

To install on Debian-based Linux distributions, such as Ubuntu, download the public GPG key and add the conda repository to the sources list.

```
# Install our public GPG key to trusted store
curl https://repo.anaconda.com/pkgs/misc/gpgkeys/anaconda.asc | gpg --dearmor > conda.gpg
install -o root -g root -m 644 conda.gpg /usr/share/keyrings/conda-archive-keyring.gpg
```

(continues on next page)

(continued from previous page)

```
# Check whether fingerprint is correct (will output an error message otherwise)
gpg --keyring /usr/share/keyrings/conda-archive-keyring.gpg --no-default-keyring --
↪fingerprint 34161F5BF5EB1D4BFBBB8F0A8AEB4F8B29D82806

# Add our Debian repo
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/conda-archive-keyring.gpg] https://
↪repo.anaconda.com/pkgs/misc/debrepo/conda stable main" > /etc/apt/sources.list.d/conda.
↪list

**NB:** If you receive a Permission denied error when trying to run the above command␣
↪(because `/etc/apt/sources.list.d/conda.list` is write protected), try using the␣
↪following command instead:
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/conda-archive-keyring.gpg] https://
↪repo.anaconda.com/pkgs/misc/debrepo/conda stable main" | sudo tee -a /etc/apt/sources.
↪list.d/conda.list
```

Conda is ready to install on your Debian-based distribution.

```
# Install it!
apt update
apt install conda
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
conda
0 upgraded, 1 newly installed, 0 to remove and 3 not upgraded.
Need to get 76.3 MB of archives.
After this operation, 221 MB of additional disk space will be used.
Get:1 https://repo.anaconda.com/pkgs/misc/debrepo/conda stable/main amd64
conda amd64 4.5.11-0 [76.3 MB]
Fetched 76.3 MB in 10s (7733 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package conda.
(Reading database ... 4799 files and directories currently installed.)
Preparing to unpack .../conda_4.5.11-0_amd64.deb ...
Unpacking conda (4.5.11-0) ...
Setting up conda (4.5.11-0) ...
```

Check to see if the installation is successful by typing:

```
source /opt/conda/etc/profile.d/conda.sh
conda -V
conda 4.5.11
```

Installing conda packages with the system package manager makes it very easy to distribute conda across a cluster of machines running Linux without having to worry about any non-privileged user modifying the installation. Any non-privileged user simply needs to run `source /opt/conda/etc/profile.d/conda.sh` to use conda.

Administrators can also distribute a .condarc file at /opt/conda/.condarc so that a predefined configuration for channels, package cache directory, and environment locations is pre-seeded to all users in a large organization. A sample configuration could look like:

```
channels:
  - defaults
pkg_dirs:
  - /shared/conda/pkgs
  - $HOME/.conda/pkgs
envs_dirs:
  - /shared/conda/envs
  - $HOME/.conda/envs
```

These RPM and Debian packages also provide another way to set up conda inside a Docker container.

**Tip**

It is recommended to use this installation method in a read-only manner and upgrade conda using the respective package manager only.

### 4.1.3 Tasks

#### Managing conda

#### Verifying that conda is installed

To verify that conda is installed, in your terminal window, run:

```
conda --version
```

Conda responds with the version number that you have installed, such as `conda 4.12.0`.

If you get an error message, make sure of the following:

- You are logged into the same user account that you used to install Anaconda or Miniconda.
- You are in a directory that Anaconda or Miniconda can find.
- You have closed and re-opened the terminal window after installing conda.

#### Determining your conda version

In addition to the `conda --version` command explained above, you can determine what conda version is installed by running one of the following commands in your terminal window:

```
conda info
```

OR

```
conda -V
```

### Updating conda to the current version

To update conda, in your terminal window, run:

```
conda update conda
```

Conda compares versions and reports what is available to install. It also tells you about other packages that will be automatically updated or changed with the update. If conda reports that a newer version is available, type `y` to update:

```
Proceed ([y]/n)? y
```

### Suppressing warning message about updating conda

To suppress the following warning message when you do not want to update conda to the latest version:

```
==> WARNING: A newer version of conda exists. <==
current version: 4.6.13
latest version: 4.8.0
```

Update conda by running: `conda update -n base conda`

Run the following command from your terminal: `conda config --set notify_outdated_conda false`

Or add the following line in your `.condarc` file: `notify_outdated_conda:  false`

### Managing environments

With conda, you can create, export, list, remove, and update environments that have different versions of Python and/or packages installed in them. Switching or moving between environments is called activating the environment. You can also share an environment file.

There are many options available for the commands described on this page. For a detailed reference on all available commands, see *commands*.

### Creating an environment with commands

Use the terminal for the following steps:

1. To create an environment:

   ```
   conda create --name <my-env>
   ```

   Replace `<my-env>` with the name of your environment.

2. When conda asks you to proceed, type `y`:

   ```
   proceed ([y]/n)?
   ```

   This creates the myenv environment in `/envs/`. No packages will be installed in this environment.

3. To create an environment with a specific version of Python:

   ```
   conda create -n myenv python=3.9
   ```

4. To create an environment with a specific package:

```
conda create -n myenv scipy
```

or:

```
conda create -n myenv python
conda install -n myenv scipy
```

5. To create an environment with a specific version of a package:

```
conda create -n myenv scipy=0.17.3
```

or:

```
conda create -n myenv python
conda install -n myenv scipy=0.17.3
```

6. To create an environment with a specific version of Python and multiple packages:

```
conda create -n myenv python=3.9 scipy=0.17.3 astroid babel
```

> **Tip**
>
> Install all the programs that you want in this environment at the same time. Installing one program at a time can lead to dependency conflicts.

To automatically install pip or another program every time a new environment is created, add the default programs to the *create_default_packages* section of your `.condarc` configuration file. The default packages are installed every time you create a new environment. If you do not want the default packages installed in a particular environment, use the `--no-default-packages` flag:

```
conda create --no-default-packages -n myenv python
```

> **Tip**
>
> You can add much more to the `conda create` command. For details, run `conda create --help`.

### Creating an environment from an environment.yml file

Use the terminal for the following steps:

1. Create the environment from the `environment.yml` file:

```
conda env create -f environment.yml
```

The first line of the `yml` file sets the new environment's name. For details see *Creating an environment file manually*.

2. Activate the new environment: `conda activate myenv`
3. Verify that the new environment was installed correctly:

```
conda env list
```

You can also use `conda info --envs`.

### Specifying a different target platform for an environment

By default, `conda` will create environments targeting the platform it's currently running on. You can check which platform you are currently on by running `conda info` and checking the `platform` entry.

However, in some cases you might want to create an environment for a different target platform or architecture. To do so, use the `--platform` flag available in the `conda create` and `conda env create` commands. See `--subdir, --platform` in *conda create* for more information about allowed values.

For example, a user running macOS on the Apple Silicon platform might want to create a `python` environment for Intel processors and emulate the executables with Rosetta. The command would be:

```
conda create --platform osx-64 --name python-x64 python
```

**Note**

You can't specify the `--platform` flag for existing environments. When created, the environment will be annotated with the custom configuration and subsequent operations on it will remember the target platform.

This flag also allows specifying a different OS (e.g. creating a Linux environment on macOS), but we don't recommend its usage outside of `--dry-run` operations. Common problems with mismatched OSes include:

- The environment cannot be solved because virtual packages are missing. You can workaround this issue by exporting the necessary `CONDA_OVERRIDE_****` environment variables. For example, solving for Linux from macOS, you will probably need `CONDA_OVERRIDE_LINUX=1` and `CONDA_OVERRIDE_GLIBC=2.17`.
- The environment can be solved, but extraction and linking fails due filesystem limitations (case insensitive systems, incompatible paths, etc). The only workaround here is to use `--dry-run --json` to obtain the solution and process the payload into a lockfile that can be shared with the target machine. See *Create explicit lockfiles without creating an environment* for more details.

### Specifying a location for an environment

You can control where a conda environment lives by providing a path to a target directory when creating the environment. For example, the following command will create a new environment in a subdirectory of the current working directory called `envs`:

```
conda create --prefix ./envs jupyterlab=3.2 matplotlib=3.5 numpy=1.21
```

You then activate an environment created with a prefix using the same command used to activate environments created by name:

```
conda activate ./envs
```

Specifying a path to a subdirectory of your project directory when creating an environment has the following benefits:

- It makes it easy to tell if your project uses an isolated environment by including the environment as a subdirectory.
- It makes your project more self-contained as everything, including the required software, is contained in a single project directory.

An additional benefit of creating your project's environment inside a subdirectory is that you can then use the same name for all your environments. If you keep all of your environments in your `envs` folder, you'll have to give each environment a different name.

There are a few things to be aware of when placing conda environments outside of the default `envs` folder.

1. Conda can no longer find your environment with the `--name` flag. You'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.
2. Specifying an install path when creating your conda environments makes it so that your command prompt is now prefixed with the active environment's absolute path rather than the environment's name.

After activating an environment using its prefix, your prompt will look similar to the following:

```
(/absolute/path/to/envs) $
```

This can result in long prefixes:

```
(/Users/USER_NAME/research/data-science/PROJECT_NAME/envs) $
```

To remove this long prefix in your shell prompt, modify the env_prompt setting in your `.condarc` file:

conda config --set env_prompt '({name})'

This will edit your `.condarc` file if you already have one or create a `.condarc` file if you do not.

Now your command prompt will display the active environment's generic name, which is the name of the environment's root folder:

```
$ cd project-directory
$ conda activate ./env
(env) project-directory $
```

### Updating an environment

You may need to update your environment for a variety of reasons. For example, it may be the case that:

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).
- you have found a better package and no longer need the older package (add new dependency and remove old dependency).

If any of these occur, all you need to do is update the contents of your `environment.yml` file accordingly and then run the following command:

```
conda env update --file environment.yml  --prune
```

**Note**

The `--prune` option causes conda to remove any dependencies that are no longer required from the environment.

### Freezing or locking an environment

CEP 22 introduced an environment marker file that will instruct `conda` not to allow modifications in the given environment. When attempting to add, update or remove a package, users will receive an error by default:

```
EnvironmentIsFrozenError: Cannot not modify '~/.conda/envs/my-env'.
The environment is marked as frozen. You can ignore this error with
the `--override-frozen` flag, at your own risk.
```

As mentioned above, users can pass the `--override-frozen` flag, but this is really not recommended and should only be done by advanced users who are aware of the risks and would have the knowledge to fix potential complications derived from that operation.

### Cloning an environment

Use the terminal for the following steps:

You can make an exact copy of an environment by creating a clone of it:

```
conda create --name myclone --clone myenv
```

**Note**

Replace `myclone` with the name of the new environment. Replace `myenv` with the name of the existing environment that you want to copy.

To verify that the copy was made:

```
conda info --envs
```

In the environments list that displays, you should see both the source environment and the new copy.

### Building identical conda environments

You can use explicit specification files to build an identical conda environment on the same operating system platform, either on the same machine or on a different machine.

Use the terminal for the following steps:

1. Run `conda list --explicit` to produce a spec list such as:

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: osx-64
@EXPLICIT
https://repo.anaconda.com/pkgs/free/osx-64/mkl-11.3.3-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/numpy-1.11.1-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/openssl-1.0.2h-1.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/pip-8.1.2-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/python-3.5.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/readline-6.2-2.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/setuptools-25.1.6-py35_0.tar.bz2
```

(continues on next page)

(continued from previous page)

```
https://repo.anaconda.com/pkgs/free/osx-64/sqlite-3.13.0-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/tk-8.5.18-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/wheel-0.29.0-py35_0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/xz-5.2.2-0.tar.bz2
https://repo.anaconda.com/pkgs/free/osx-64/zlib-1.2.8-3.tar.bz2
```

2. To create this spec list as a file in the current working directory, run:

```
conda list --explicit > spec-file.txt
```

> **Note**
>
> You can use `spec-file.txt` as the filename or replace it with a filename of your choice.

An explicit spec file is not usually cross platform, and therefore has a comment at the top such as `# platform: osx-64` showing the platform where it was created. This platform is the one where this spec file is known to work. On other platforms, the packages specified might not be available or dependencies might be missing for some of the key packages already in the spec.

To use the spec file to create an identical environment on the same machine or another machine:

```
conda create --name myenv --file spec-file.txt
```

To use the spec file to install its listed packages into an existing environment:

```
conda install --name myenv --file spec-file.txt
```

Conda does not check architecture or dependencies when installing from a spec file. To ensure that the packages work correctly, make sure that the file was created from a working environment, and use it on the same architecture, operating system, and platform, such as linux-64 or osx-64.

### Activating an environment

Activating environments is essential to making the software in the environments work well. Activation entails two primary functions: adding entries to PATH for the environment and running any activation scripts that the environment may contain. These activation scripts are how packages can set arbitrary environment variables that may be necessary for their operation. You can also *use the config API to set environment variables*.

Activation prepends to PATH. This only takes effect when you have the environment active so it is local to a terminal session, not global.

> **Note**
>
> When installing Anaconda, you have the option to "Add Anaconda to my PATH environment variable." *This is not recommended* because it *appends* Anaconda to PATH. When the installer appends to PATH, it does not call the activation scripts.

> **Note**

On Windows, PATH is composed of two parts, the *system* PATH and the *user* PATH. The system PATH always comes first. When you install Anaconda for "Just Me", we add it to the *user* PATH. When you install for "All Users", we add it to the *system* PATH. In the former case, you can end up with system PATH values taking precedence over your entries. In the latter case, you do not. *We do not recommend* multi-user installs.

To activate an environment: `conda activate myenv`

**Note**

Replace `myenv` with the environment name or directory path.

Conda prepends the path name `myenv` onto your system command.

You may receive a warning message if you have not activated your environment:

```
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation.
```

If you receive this warning, you need to activate your environment. To do so on Windows, run: `c:\Anaconda3\Scripts\activate` in a terminal window.

Windows is extremely sensitive to proper activation. This is because the Windows library loader does not support the concept of libraries and executables that know where to search for their dependencies (RPATH). Instead, Windows relies on a dynamic-link library search order.

If environments are not active, libraries won't be found and there will be lots of errors. HTTP or SSL errors are common errors when the Python in a child environment can't find the necessary OpenSSL library.

Conda itself includes some special workarounds to add its necessary PATH entries. This makes it so that it can be called without activation or with any child environment active. In general, calling any executable in an environment without first activating that environment will likely not work. For the ability to run executables in activated environments, you may be interested in the `conda run` command.

If you experience errors with PATH, review our *troubleshooting*.

### Conda init

Earlier versions of conda introduced scripts to make activation behavior uniform across operating systems. Conda 4.4 allowed `conda activate myenv`. Conda 4.6 added extensive initialization support so that conda works faster and less disruptively on a wide variety of shells (bash, zsh, csh, fish, xonsh, and more). Now these shells can use the `conda activate` command.

Alternatively, `conda init --condabin` will not install a shell function in your profile. Instead, it will only add the `$CONDA_PREFIX/condabin/` directory to `PATH`. This directory only contains the `conda` executable, so it should be minimally invasive.

For more information, read the output from `conda init --help`.

One setting may be useful to you when using `conda init` is:

```
auto_activate: bool
```

This setting controls whether or not conda activates the given default environment when it first starts up. You'll have the `conda` command available either way, but without activating the environment, none of the other programs in the default environment will be available until the environment is activated with `conda activate`.

The environment to be activated by default can be configured with:

```
default_activation_env: str
```

People sometimes choose this setting to speed up the time their shell takes to start up or to keep conda-installed software from automatically hiding their other software.

### Nested activation

By default, `conda activate` will deactivate the current environment before activating the new environment and re-activate it when deactivating the new environment. Sometimes you may want to leave the current environment PATH entries in place so that you can continue to easily access command-line programs from the first environment. This is most commonly encountered when common command-line utilities are installed in the default environment. To retain the current environment in the PATH, you can activate the new environment using:

```
conda activate --stack myenv
```

If you wish to always stack when going from the outermost environment, which is typically the default environment, you can set the `auto_stack` configuration option:

```
conda config --set auto_stack 1
```

You may specify a larger number for a deeper level of automatic stacking, but this is not recommended since deeper levels of stacking are more likely to lead to confusion.

### Environment variable for DLL loading verification

If you don't want to activate your environment and you want Python to work for DLL loading verification, then follow the *troubleshooting directions*.

**Warning**

If you choose not to activate your environment, then loading and setting environment variables to activate scripts will not happen. We only support activation.

### Deactivating an environment

To deactivate an environment, type: `conda deactivate`

Conda removes the path name for the currently active environment from your system command.

**Note**

To simply return to the default environment, it's better to call `conda activate` with no environment specified, rather than to try to deactivate. If you run `conda deactivate` from your base environment, you may lose the ability to run conda at all. Don't worry, that's local to this shell - you can start a new one. However, if the environment was activated using `--stack` (or was automatically stacked) then it is better to use `conda deactivate`.

### Determining your current environment

Use the terminal for the following steps.

By default, the active environment---the one you are currently using---is shown in parentheses () or brackets [] at the beginning of your command prompt:

```
(myenv) $
```

If you do not see this, run:

```
conda info --envs
```

In the environments list that displays, your current environment is highlighted with an asterisk (*).

By default, the command prompt is set to show the name of the active environment. To disable this option:

```
conda config --set changeps1 false
```

To re-enable this option:

```
conda config --set changeps1 true
```

### Viewing a list of your environments

To see a list of all of your environments, in your terminal window, run:

```
conda info --envs
```

OR

```
conda env list
```

A list similar to the following is displayed:

```
conda environments:
myenv                  /home/username/miniconda/envs/myenv
snowflakes             /home/username/miniconda/envs/snowflakes
bunnies                /home/username/miniconda/envs/bunnies
```

If this command is run by an administrator, a list of all environments belonging to all users will be displayed.

### Viewing a list of the packages in an environment

To see a list of all packages installed in a specific environment:

- If the environment is not activated, in your terminal window, run:

  ```
  conda list -n myenv
  ```

- If the environment is activated, in your terminal window, run:

  ```
  conda list
  ```

- To see if a specific package is installed in an environment, in your terminal window, run:

```
conda list -n myenv scipy
```

### Using pip in an environment

To use pip in your environment, in your terminal window, run:

```
conda install -n myenv pip
conda activate myenv
pip <pip_subcommand>
```

Issues may arise when using pip and conda together. When combining conda and pip, it is best to use an isolated conda environment. Only after conda has been used to install as many packages as possible should pip be used to install any remaining software. If modifications are needed to the environment, it is best to create a new environment rather than running conda after pip. When appropriate, conda and pip requirements should be stored in text files.

We recommend that you:

**Use pip only after conda**

- Install as many requirements as possible with conda then use pip.
- Pip should be run with `--upgrade-strategy only-if-needed` (the default).
- Do not use pip with the `--user` argument, avoid all users installs.

**Use conda environments for isolation**

- Create a conda environment to isolate any changes pip makes.
- Environments take up little space thanks to hard links.
- Care should be taken to avoid running pip in the root environment.

**Recreate the environment if changes are needed**

- Once pip has been used, conda will be unaware of the changes.
- To install additional conda packages, it is best to recreate the environment.

**Store conda and pip requirements in text files**

- Package requirements can be passed to conda via the `--file` argument.
- Pip accepts a list of Python packages with `-r` or `--requirements`.
- Conda env will export or create environments based on a file with conda and pip requirements.

### Setting environment variables

If you want to associate environment variables with an environment, you can use the config API. This is recommended as an alternative to using activate and deactivate scripts since those are an execution of arbitrary code that may not be safe.

First, create your environment and activate it:

```
conda create -n test-env
conda activate test-env
```

To list any variables you may have, run `conda env config vars list`.

To set environment variables, run `conda env config vars set my_var=value`.

Once you have set an environment variable, you have to reactivate your environment: `conda activate test-env`.

To check if the environment variable has been set, run `echo $my_var` (`echo %my_var%` on Windows) or `conda env config vars list`.

When you deactivate your environment, you can use those same commands to see that the environment variable goes away.

You can specify the environment you want to affect using the `-n` and `-p` flags. The `-n` flag allows you to name the environment and `-p` allows you to specify the path to the environment.

To unset the environment variable, run `conda env config vars unset my_var -n test-env`.

When you deactivate your environment, you can see that environment variable goes away by rerunning `echo my_var` or `conda env config vars list` to show that the variable name is no longer present.

Environment variables set using `conda env config vars` will be retained in the output of `conda env export`. Further, you can declare environment variables in the environment.yml file as shown here:

```
name: env-name
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.7
  - codecov
variables:
  VAR1: valueA
  VAR2: valueB
```

### Saving environment variables

Conda environments can include saved environment variables.

Suppose you want an environment "analytics" to store both a secret key needed to log in to a server and a path to a configuration file. The sections below explain how to write a script named `env_vars` to do this on *Windows* and *macOS or Linux*.

This type of script file can be part of a conda package, in which case these environment variables become active when an environment containing that package is activated.

You can name these scripts anything you like. However, multiple packages may create script files, so be sure to use descriptive names that are not used by other packages. One popular option is to give the script a name in the form `packagename-scriptname.sh`, or on Windows, `packagename-scriptname.bat`.

### Windows

1. Locate the directory for the conda environment in your terminal window by running in the command shell `%CONDA_PREFIX%`.
2. Enter that directory and create these subdirectories and files:

```
cd %CONDA_PREFIX%
mkdir .\etc\conda\activate.d
mkdir .\etc\conda\deactivate.d
type NUL > .\etc\conda\activate.d\env_vars.bat
type NUL > .\etc\conda\deactivate.d\env_vars.bat
```

3. Edit `.\etc\conda\activate.d\env_vars.bat` as follows:

```
set MY_KEY='secret-key-value'
set MY_FILE=C:\path\to\my\file
```

4. Edit `.\etc\conda\deactivate.d\env_vars.bat` as follows:

```
set MY_KEY=
set MY_FILE=
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

### macOS and Linux

1. Locate the directory for the conda environment in your terminal window by running in the terminal `echo $CONDA_PREFIX`.
2. Enter that directory and create these subdirectories and files:

```
cd $CONDA_PREFIX
mkdir -p ./etc/conda/activate.d
mkdir -p ./etc/conda/deactivate.d
touch ./etc/conda/activate.d/env_vars.sh
touch ./etc/conda/deactivate.d/env_vars.sh
```

3. Edit `./etc/conda/activate.d/env_vars.sh` as follows:

```
#!/bin/sh

export MY_KEY='secret-key-value'
export MY_FILE=/path/to/my/file/
```

4. Edit `./etc/conda/deactivate.d/env_vars.sh` as follows:

```
#!/bin/sh

unset MY_KEY
unset MY_FILE
```

When you run `conda activate analytics`, the environment variables `MY_KEY` and `MY_FILE` are set to the values you wrote into the file. When you run `conda deactivate`, those variables are erased.

### Sharing an environment

You may want to share your environment with someone else---for example, so they can re-create a test that you have done. To allow them to quickly reproduce your environment, with all of its packages and versions, give them a copy of your `environment.yml` file.

### Exporting the environment.yml file

> **Note**
>
> If you already have an `environment.yml` file in your current directory, it will be overwritten during this task.

1. Activate the environment to export: `conda activate myenv`

   > **Note**
   >
   > Replace `myenv` with the name of the environment.

2. Export your active environment to a new file:

   ```
   conda env export > environment.yml
   ```

   > **Note**
   >
   > This file handles both the environment's pip packages and conda packages.

3. Email or copy the exported `environment.yml` file to the other person.

### Exporting an environment file across platforms

If you want to make your environment file work across platforms, you can use the `conda env export --from-history` flag. This will only include packages that you've explicitly asked for, as opposed to including every package in your environment.

For example, if you create an environment and install Python and a package:

```
conda install python=3.7 codecov
```

This will download and install numerous additional packages to solve for dependencies. This will introduce packages that may not be compatible across platforms.

If you use `conda env export`, it will export all of those packages. However, if you use `conda env export --from-history`, it will only export those you specifically chose:

```
(env-name)   ~ conda env export --from-history
name: env-name
channels:
  - conda-forge
```

(continues on next page)

(continued from previous page)

```
  - defaults
dependencies:
  - python=3.7
  - codecov
prefix: /Users/username/anaconda3/envs/env-name
```

**Note**

If you installed Anaconda 2019.10 on macOS, your prefix may be `/Users/username/opt/envs/env-name`.

### Creating an environment file manually

You can create an environment file (`environment.yml`) manually to share with others.

EXAMPLE: A simple environment file:

```
name: stats
dependencies:
  - numpy
  - pandas
```

EXAMPLE: A more complex environment file:

```
name: stats2
channels:
  - javascript
dependencies:
  - python=3.9
  - bokeh=2.4.2
  - conda-forge::numpy=1.21.*
  - nodejs=16.13.*
  - flask
  - pip
  - pip:
    - Flask-Testing
```

**Note**

**Using wildcards**

Note the use of the wildcard * when defining a few of the versions in the complex environment file. Keeping the major and minor versions fixed while allowing the patch to be any number allows you to use your environment file to get any bug fixes while still maintaining consistency in your environment. For more information on package installation values, see *Package search and install specifications*.

**Specifying channels outside of "channels"**

You may occasionally want to specify which channel conda will use to install a specific package. To accomplish this, use the *channel::package* syntax in *dependencies:*, as demonstrated above with *conda-forge::numpy* (version numbers optional). The specified channel does not need to be present in the *channels:* list, which is useful if you want some—but not *all*—packages installed from a community channel such as *conda-forge*.

You can exclude the default channels by adding `nodefaults` to the channels list.

```
channels:
  - javascript
  - nodefaults
```

This is equivalent to passing the `--override-channels` option to most `conda` commands.

Adding `nodefaults` to the channels list in `environment.yml` is similar to removing `defaults` from the *channels list* in the `.condarc` file. However, changing `environment.yml` affects only one of your conda environments while changing `.condarc` affects them all.

For details on creating an environment from this `environment.yml` file, see *Creating an environment from an environment.yml file*.

### Restoring an environment

Conda keeps a history of all the changes made to your environment, so you can easily "roll back" to a previous version. To list the history of each change to the current environment: `conda list --revisions`

To restore environment to a previous revision: `conda install --revision=REVNUM` or `conda install --rev REVNUM`.

**Note**

Replace REVNUM with the revision number.

Example: If you want to restore your environment to revision 8, run `conda install --rev 8`.

### Removing an environment

To remove an environment, in your terminal window, run:

```
conda remove --name myenv --all
```

You may instead use `conda env remove --name myenv`.

To verify that the environment was removed, in your terminal window, run:

```
conda info --envs
```

The environments list that displays should not show the removed environment.

### Create explicit lockfiles without creating an environment

`@EXPLICIT` lockfiles allow you to (re)create environments without invoking the solver. They consist of an `@EXPLICIT` header plus a list of conda package URLs, optionally followed by their MD5 or SHA256 hash.

They can be obtained from existing environments via `conda list --explicit`, as seen in *Building identical conda environments*.

But what if you only need the lockfile? Would you need create to a temporary environment first just to delete it later? Fortunately, there's a way: you can invoke `conda` in JSON mode and then process the output with `jq`.

> **Tip**
>
> You'll need `jq` in your system. If you don't have it yet, you can install it via `conda` (e.g. `conda create -n jq jq`) or via your system package manager.

The command looks like this for Linux and macOS (replace `MATCHSPECS_GO_HERE` with the relevant packages you want):

```
echo "@EXPLICIT" > explicit.txt
CONDA_PKGS_DIRS=$(mktemp -d) conda create --dry-run MATCHSPECS_GO_HERE --json | jq -r '.
↪actions.FETCH[] | .url + "#" + .md5' >> explicit.txt
```

The syntax in Windows only needs some small changes:

```
echo "@EXPLICIT" > explicit.txt
set "CONDA_PKGS_DIRS=%TMP%\conda-%RANDOM%"
conda create --dry-run MATCHSPECS_GO_HERE --json | jq -r '.actions.FETCH[] | .url + "#"␣
↪+ .md5' >> explicit.txt
set "CONDA_PKGS_DIRS="
```

The resulting `explicit.txt` can be used to create a new environment with:

```
conda create -n new-environment --file explicit.txt
```

### Managing channels

Conda channels are the locations where packages are stored. They serve as the base for hosting and managing packages. Conda packages are downloaded from remote channels, which are URLs to directories containing conda packages. The conda command searches a default set of channels and packages are automatically downloaded and updated from the default channel. Read more about *conda channels* and the various terms of service for their use.

Different channels can have the same package, so conda must handle these channel collisions.

There will be no channel collisions if you use only the defaults channel. There will also be no channel collisions if all of the channels you use only contain packages that do not exist in any of the other channels in your list. The way conda resolves these collisions matters only when you have multiple channels in your channel list that host the same package.

By default, conda prefers packages from a higher priority channel over any version from a lower priority channel. Therefore, you can now safely put channels at the bottom of your channel list to provide additional packages that are not in the default channels and still be confident that these channels will not override the core package set.

Conda collects all of the packages with the same name across all listed channels and processes them as follows:

1. Sorts packages from highest to lowest channel priority.
2. Sorts tied packages---packages with the same channel priority---from highest to lowest version number. For example, if channelA contains NumPy 1.12.0 and 1.13.1, NumPy 1.13.1 will be sorted higher.
3. Sorts still-tied packages---packages with the same channel priority and same version---from highest to lowest build number. For example, if channelA contains both NumPy 1.12.0 build 1 and build 2, build 2 is sorted first. Any packages in channelB would be sorted below those in channelA.
4. Installs the first package on the sorted list that satisfies the installation specifications.

Essentially, the order goes: channelA::numpy-1.13_1 > channelA::numpy-1.12.1_1 > channelA::numpy-1.12.1_0 > channelB::numpy-1.13_1

> **Note**
>
> If strict channel priority is turned on then channelB::numpy-1.13_1 isn't included in the list at all.

To make conda install the newest version of a package in any listed channel:

- Add `channel_priority: disabled` to your `.condarc` file.

  OR

- Run the equivalent command:

```
conda config --set channel_priority disabled
```

Conda then sorts as follows:

1. Sorts the package list from highest to lowest version number.
2. Sorts tied packages from highest to lowest channel priority.
3. Sorts tied packages from highest to lowest build number.

Because build numbers from different channels are not comparable, build number still comes after channel priority.

The following command adds the channel "new_channel" to the top of the channel list, making it the highest priority:

```
conda config --add channels new_channel
```

Conda has an equivalent command:

```
conda config --prepend channels new_channel
```

Conda also has a command that adds the new channel to the bottom of the channel list, making it the lowest priority:

```
conda config --append channels new_channel
```

### Strict channel priority

As of version 4.6.0, Conda has a strict channel priority feature. Strict channel priority can dramatically speed up conda operations and also reduce package incompatibility problems. We recommend setting channel priority to "strict" when possible.

Details about it can be seen by typing `conda config --describe channel_priority`.

```
channel_priority (ChannelPriority)
Accepts values of 'strict', 'flexible', and 'disabled'. The default
value is 'flexible'. With strict channel priority, packages in lower
priority channels are not considered if a package with the same name
appears in a higher priority channel. With flexible channel priority,
the solver may reach into lower priority channels to fulfill
dependencies, rather than raising an unsatisfiable error. With channel
priority disabled, package version takes precedence, and the
configured priority of channels is used only to break ties. In
previous versions of conda, this parameter was configured as either
True or False. True is now an alias to 'flexible'.

channel_priority: flexible
```

### Managing packages

> **Note**
>
> There are many options available for the commands described on this page. For details, see *commands*.

### Searching for packages

Use the terminal for the following steps.

To see if a specific package, such as SciPy, is available for installation:

```
conda search scipy
```

To see if a specific package, such as SciPy, is available for installation from Anaconda.org:

```
conda search --override-channels --channel defaults scipy
```

To see if a specific package, such as iminuit, exists in a specific channel, such as http://conda.anaconda.org/mutirri, and is available for installation:

```
conda search --override-channels --channel http://conda.anaconda.org/mutirri iminuit
```

### Installing packages

Use the terminal for the following steps.

To install a specific package such as SciPy into an existing environment "myenv":

```
conda install --name myenv scipy
```

If you do not specify the environment name, which in this example is done by `--name myenv`, the package installs into the current environment:

```
conda install scipy
```

To install a specific version of a package such as SciPy:

```
conda install scipy=0.15.0
```

To install multiple packages at once, such as SciPy and cURL:

```
conda install scipy curl
```

> **Note**
>
> It is best to install all packages at once, so that all of the dependencies are installed at the same time.

To install multiple packages at once and specify the version of the package:

```
conda install scipy=0.15.0 curl=7.26.0
```

To install a package for a specific Python version:

```
conda install scipy=0.15.0 curl=7.26.0 -n py34_env
```

If you want to use a specific Python version, it is best to use an environment with that version. For more information, see *Troubleshooting*.

### Installing similar packages

Installing packages that have similar filenames and serve similar purposes may return unexpected results. The package last installed will likely determine the outcome, which may be undesirable. If the two packages have different names, or if you're building variants of packages and need to line up other software in the stack, we recommend using *Mutex metapackages*.

### Installing packages from Anaconda.org

Packages that are not available using `conda install` can be obtained from Anaconda.org, a package management service for both public and private package repositories. Anaconda.org is an Anaconda product, just like Anaconda and Miniconda.

To install a package from Anaconda.org:

1. In a browser, go to http://anaconda.org.
2. To find the package named bottleneck, type `bottleneck` in the top-left box named Search Packages.
3. Find the package that you want and click it to go to the detail page.

   The detail page displays the name of the channel. In this example it is the "pandas" channel.
4. Now that you know the channel name, use the `conda install` command to install the package. In your terminal window, run:

   ```
   conda install -c pandas bottleneck
   ```

   This command tells conda to install the bottleneck package from the pandas channel on Anaconda.org.
5. To check that the package is installed, in your terminal window, run:

   ```
   conda list
   ```

   A list of packages appears, including bottleneck.

> **Note**
>
> For information on installing packages from multiple channels, see *Managing channels*.

### Installing non-conda packages

If a package is not available from conda or Anaconda.org, you may be able to find and install the package via conda-forge or with another package manager like pip.

Pip packages do not have all the features of conda packages and we recommend first trying to install any package with conda. If the package is unavailable through conda, try finding and installing it with conda-forge.

If you still cannot install the package, you can try installing it with pip. The differences between pip and conda packages cause certain unavoidable limits in compatibility but conda works hard to be as compatible with pip as possible.

> **Note**
>
> Both pip and conda are included in Anaconda and Miniconda, so you do not need to install them separately.
>
> Conda environments replace virtualenv, so there is no need to activate a virtualenv before using pip.

It is possible to have pip installed outside a conda environment or inside a conda environment.

To gain the benefits of conda integration, be sure to install pip inside the currently active conda environment and then install packages with that instance of pip. The command `conda list` shows packages installed this way, with a label showing that they were installed with pip.

You can install pip in the current conda environment with the command `conda install pip`, as discussed in *Using pip in an environment*.

If there are instances of pip installed both inside and outside the current conda environment, the instance of pip installed inside the current conda environment is used.

To install a non-conda package:

1. Activate the environment where you want to put the program:
   - In your terminal window, run `conda activate myenv`.
2. To use pip to install a program such as See, in your terminal window, run:

   ```
   pip install see
   ```

3. To verify the package was installed, in your terminal window, run:

   ```
   conda list
   ```

   If the package is not shown, install pip as described in *Using pip in an environment* and try these commands again.

### Installing commercial packages

Installing a commercial package such as IOPro is the same as installing any other package. In your terminal window, run:

```
conda install --name myenv iopro
```

This command installs a free trial of one of Anaconda's commercial packages called IOPro, which can speed up your Python processing. Except for academic use, this free trial expires after 30 days.

### Viewing a list of installed packages

Use the terminal for the following steps.

To list all of the packages in the active environment:

```
conda list
```

To list all of the packages in a deactivated environment:

```
conda list -n myenv
```

### Listing package dependencies

To find what packages are depending on a specific package in your environment, there is not one specific conda command. It requires a series of steps:

1. List the dependencies that a specific package requires to run: `conda search package_name --info`
2. Find your installation's package cache directory: `conda info`
3. Find package dependencies. By default, Anaconda/Miniconda stores packages in ~/anaconda/pkgs/ (or ~/opt/pkgs/ on macOS Catalina). Each package has an index.json file which lists the package's dependencies. This file resides in ~anaconda/pkgs/package_name/info/index.json.
4. Now you can find what packages depend on a specific package. Use grep to search all index.json files as follows: `grep package_name ~/anaconda/pkgs/*/info/index.json`

The result will be the full package path and version of anything containing the <package_name>.

Example: `grep numpy ~/anaconda3/pkgs/*/info/index.json`

Output from the above command:

```
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpy 1.11.3␣
↪py36_0
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpydoc 0.6.
↪0 py36_0
/Users/testuser/anaconda3/pkgs/anaconda-4.3.0-np111py36_0/info/index.json: numpy 1.11.3␣
↪py36_0
```

Note this also returned "numpydoc" as it contains the string "numpy". To get a more specific result set you can add < and >.

### Updating packages

Use `conda update` command to check to see if a new update is available. If conda tells you an update is available, you can then choose whether or not to install it.

Use the terminal for the following steps.

- To update a specific package:

  ```
  conda update biopython
  ```

- To update Python:

```
conda update python
```

- To update conda itself:

```
conda update conda
```

> **Note**
>
> Conda updates to the highest version in its series, so Python 3.9 updates to the highest available in the 3.x series.

To update the Anaconda metapackage:

```
conda update conda
conda update anaconda
```

Regardless of what package you are updating, conda compares versions and then reports what is available to install. If no updates are available, conda reports "All requested packages are already installed."

If a newer version of your package is available and you wish to update it, type `y` to update:

```
Proceed ([y]/n)? y
```

### Preventing packages from updating (pinning)

Pinning a package specification in an environment prevents packages listed in the `pinned` file from being updated.

In the environment's `conda-meta` directory, add a file named `pinned` that includes a list of the packages that you do not want updated.

EXAMPLE: The file below forces NumPy to stay on the 1.7 series, which is any version that starts with 1.7. This also forces SciPy to stay at exactly version 0.14.2:

```
numpy 1.7.*
scipy ==0.14.2
```

With this `pinned` file, `conda update numpy` keeps NumPy at 1.7.1, and `conda install scipy=0.15.0` causes an error.

Use the `--no-pin` flag to override the update restriction on a package. In the terminal, run:

```
conda update numpy --no-pin
```

Because the `pinned` specs are included with each conda install, subsequent `conda update` commands without `--no-pin` will revert NumPy back to the 1.7 series.

### Adding default packages to new environments automatically

To automatically add default packages to each new environment that you create:

1. Open a terminal window and run: `conda config --add create_default_packages PACKAGENAME1 PACKAGENAME2`
2. Now, you can create new environments and the default packages will be installed in all of them.

You can also *edit the .condarc file* with a list of packages to create by default.

You can override this option at the command prompt with the `--no-default-packages` flag.

### Removing packages

Use the terminal for the following steps.

- To remove a package such as SciPy in an environment such as myenv:

```
conda remove -n myenv scipy
```

- To remove a package such as SciPy in the current environment:

```
conda remove scipy
```

- To remove multiple packages at once, such as SciPy and cURL:

```
conda remove scipy curl
```

- To confirm that a package has been removed:

```
conda list
```

### Managing Python

Conda treats Python the same as any other package, so it is easy to manage and update multiple installations.

Conda supports Python 3.9, 3.10, 3.11 and 3.12.

### Viewing a list of available Python versions

To list the versions of Python that are available to install, in your terminal window, run:

```
conda search python
```

This lists all packages whose names contain the text `python`.

To list only the packages whose full name is exactly `python`, add the `--full-name` option. In your terminal window, run:

```
conda search --full-name python
```

### Installing a different version of Python

To install a different version of Python without overwriting the current version, create a new environment and install the second Python version into it:

1. Create the new environment:
   - To create the new environment for Python 3.9, in your terminal window run:

```
conda create -n py39 python=3.9
```

> **Note**
>
> Replace `py39` with the name of the environment you want to create. `python=3.9` is the package and version you want to install in this new environment. This could be any package, such as `numpy=1.19`, or *multiple packages*.

2. *Activate the new environment*.
3. Verify that the new environment is your *current environment*.
4. To verify that the current environment uses the new Python version, in your terminal window, run:

```
python --version
```

### Installing PyPy

To use the PyPy builds you can do the following:

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda create -n pypy pypy
conda activate pypy
```

### Using a different version of Python

To switch to an environment that has different version of Python, *activate the environment*.

### Updating Python

To update Python to the latest version in your environment, run:

```
conda update python
```

This command will update you to the latest major release (e.g. from `python=3.10` to `python=3.12`).

If you would like to remain on a minor release, use the `conda install` command instead:

```
conda install python=3.10
```

### Managing virtual packages

"Virtual" packages are injected into the conda solver to allow real packages to depend on features present on the system that cannot be managed directly by conda, like system driver versions or CPU features. Virtual packages are not real packages and not displayed by `conda list`. Instead `conda` runs a small bit of code to detect the presence or absence of the system feature that corresponds to the package. The currently supported list of virtual packages includes:

- `__cuda`: Maximum version of CUDA supported by the display driver.
- `__osx`: OSX version if applicable.
- `__glibc`: Version of glibc supported by the OS.
- `__linux`: Available when running on Linux.
- `__unix`: Available when running on OSX or Linux.
- `__win`: Available when running on Win.
- `__conda`: Version of conda that is being used for solving.

Other virtual packages will be added in future conda releases. These are denoted by a leading double-underscore in the package name.

> **Note**
>
> Note that as of version `22.11.0`, *virtual packages* are implemented as *conda plugins*.

### Listing detected virtual packages

Use the terminal for the following steps.

To see the list of detected virtual packages, run:

```
conda info
```

If a package is detected, you will see it listed in the `virtual packages` section, as shown in this example:

```
     active environment : base
    active env location : /Users/demo/dev/conda/devenv
            shell level : 1
       user config file : /Users/demo/.condarc
populated config files : /Users/demo/.condarc
         conda version : 4.6.3.post8+8f640d35a
    conda-build version : 3.17.8
         python version : 3.7.2.final.0
       virtual packages : __cuda=10.0
       base environment : /Users/demo/dev/conda/devenv (writable)
           channel URLs : https://repo.anaconda.com/pkgs/main/osx-64
                          https://repo.anaconda.com/pkgs/main/noarch
                          https://repo.anaconda.com/pkgs/free/osx-64
                          https://repo.anaconda.com/pkgs/free/noarch
                          https://repo.anaconda.com/pkgs/r/osx-64
                          https://repo.anaconda.com/pkgs/r/noarch
          package cache : /Users/demo/dev/conda/devenv/pkgs
                          /Users/demo/.conda/pkgs
```

(continues on next page)

(continued from previous page)

```
     envs directories : /Users/demo/dev/conda/devenv/envs
                        /Users/demo/.conda/envs
             platform : osx-64
           user-agent : conda/4.6.3.post8+8f640d35a requests/2.21.0 CPython/3.7.2␣
↪Darwin/17.7.0 OSX/10.13.6
                UID:GID : 502:20
             netrc file : None
           offline mode : False
```

### Overriding detected packages

For troubleshooting, it is possible to override virtual package detection using an environment variable. Supported variables include:

- `CONDA_OVERRIDE_CUDA` - CUDA version number or set to "" for no CUDA detected.
- `CONDA_OVERRIDE_OSX` - OSX version number or set to "" for no OSX detected.
- `CONDA_OVERRIDE_GLIBC` - GLIBC version number or set to "" for no GLIBC. This only applies on Linux.

### Creating custom channels

In this tutorial, we walk through how to create your own channel that can either be accessed via the local or network file system or served from a webserver.

To create a custom channel:

1. You will need to install conda-build to complete this tutorial. If you do not already have it, you can install it with the following command:

   ```
   conda install conda-build
   ```

2. Organize all the packages in subdirectories for the platforms you wish to serve. Below is an example of what this may look like:

   ```
   channel
   ├── linux-64
   │   └── package-1.0-0.tar.bz2
   ├── osx-64
   │   └── package-1.0-0.tar.bz2
   └── win-64
       └── package-1.0-0.tar.bz2
   ```

3. Run `conda index` on the channel root directory:

   ```
   conda index channel/
   ```

   The conda index command generates a file `repodata.json`, saved to each repository directory, which conda uses to get the metadata for the packages in the channel.

> **Note**
>
> Each time you add or modify a package in the channel, you must rerun `conda index` for conda to see the update.

4. To test custom channels, serve the custom channel using a web server or using a `file://` URL to the channel directory. Test by sending a search command to the custom channel.

   **Example**: if you want a file in the custom channel location `/opt/channel/linux-64/`, search for files in that location:

```
conda search -c file:///opt/channel/ --override-channels
```

> **Note**
>
> - The channel URL does not include the platform, as conda automatically detects and adds the platform.
> - The option `--override-channels` ensures that conda searches only your specified channel and no other channels, such as default channels or any other channels you may have listed in your `.condarc` file.

   If you have set up your private repository correctly, you get the following output:

```
Fetching package metadata: . . . .
```

   This is followed by a list of the conda packages found. This verifies that you have set up and indexed your private repository successfully.

### Creating projects

In this tutorial, we will walk through how to set up a new Python project in conda using an `environment.yml` file. This file will help you keep track of your dependencies and share your project with others. We cover how to create your project, add a simple Python program and update it with new dependencies.

### Requirements

To follow along, you will need a working conda installation. Please head over to our *installation guide* for instructions on how to get conda installed if you do not have it.

This tutorial relies heavily on using your computer's terminal (Command Prompt or PowerShell on Windows), so it is also important to have a working familiarity with using basic commands such as `cd` and `ls`.

### Creating the project's files

To start off, we will need a directory that will contain the files for our project. This can be created with the following command:

```
mkdir my-project
```

In this directory, we will now create a new `environment.yaml` file, which will hold the dependencies for our Python project. In your text editor (e.g. VSCode, PyCharm, vim, etc.), create this file and add the following:

```
name: my-project
channels:
  - defaults
dependencies:
  - python
```

Let's briefly go over what each part of this file means.

**Name**
: The name of your environment. Here, we have chosen the name "my-project", but this can be anything you want.

**Channels**
: Channels specify where you want conda to search for packages. We have chosen the `defaults` channel, but others such as `conda-forge` or `bioconda` are also possible to list here.

**Dependencies**
: All the dependencies that you need for your project. So far, we have just added `python` because we know it will be a Python project. We will add more later.

### Creating our environment

Now that we have written a basic `environment.yml` file, we can create and activate an environment from it. To do so, run the following commands:

```
conda env create --file environment.yml
conda activate my-project
```

### Creating our Python application

With our new environment with Python installed, we can create a simple Python program. In your project folder, create a `main.py` file and add the following:

```
def main():
    print("Hello, conda!")


if __name__ == "__main__":
    main()
```

We can run our simple Python program by running the following command:

```
python main.py
Hello, conda!
```

### Updating our project with new dependencies

If you want your project to do more than the simple example above, you can use one of the thousands of available packages on conda channels. To demonstrate this, we will add a new dependency so that we can pull in some data from the internet and perform a basic analysis.

To perform the data analysis, we will be relying on the Pandas package. To add this to our project, we will need to update our `environment.yml` file:

```
name: my-project
channels:
  - defaults
dependencies:
  - python
  - pandas  # <-- This is our new dependency
```

Once we have done that, we can run the `conda env update` command to install the new package:

```
conda env update --file environment.yml
```

Now that our dependencies are installed, we will download some data to use for our analysis. For this, we will use the U.S. Environmental Protection Agency's Walkability Index dataset available on data.gov. You can download this with the following command:

```
curl -O https://edg.epa.gov/EPADataCommons/public/OA/EPA_SmartLocationDatabase_V3_Jan_
↪2021_Final.csv
```

**Tip**

If you do not have `curl`, you can visit the above link with a web browser to download it.

For our analysis, we are interested in knowing what percentage of U.S. residents live in highly walkable areas. This is a question that we can easily answer using the `pandas` library. Below is an example of how you might go about doing that:

```
import pandas as pd


def main():
    """
    Answers the question:

    What percentage of U.S. residents live highly walkable neighborhoods?

    "15.26" is the threshold on the index for a highly walkable area.
    """
    csv_file = "./EPA_SmartLocationDatabase_V3_Jan_2021_Final.csv"
    highly_walkable = 15.26

    df = pd.read_csv(csv_file)

    total_population = df["TotPop"].sum()
    highly_walkable_pop = df[df["NatWalkInd"] >= highly_walkable]["TotPop"].sum()
```

(continues on next page)

(continued from previous page)

```
    percentage = (highly_walkable_pop / total_population) * 100.0

    print(
        f"{percentage:.2f}% of U.S. residents live in highly" "walkable neighborhoods."
    )


if __name__ == "__main__":
    main()
```

Update your `main.py` file with the code above and run it. You should get the following answer:

```
python main.py
10.69% of Americans live in highly walkable neighborhoods
```

### Conclusion

You have just been introduced to creating your own data analysis project by using the `environment.yml` file in conda. As the project grows, you may wish to add more dependencies and also better organize the Python code into separate files and modules.

For even more information about working with environments and `environment.yml` files, please see *Managing Environments*.

### Viewing command-line help

To see a list of supported conda commands, in your terminal window, run:

```
conda --help
```

or

```
conda -h
```

To get help for a specific command, type the command name followed by `--help`.

**Example**

To see help for the `create` command, in your terminal window, run:

```
conda create -h
```

> **Note**
>
> You can see the same command help in *commands*.

The tasks section is organized into various pages which cover nearly everything you can do with conda.

### Common Tasks

*Managing conda*
: Everything necessary to know about managing your installation of conda

*Managing environments*
: Various operations involved with creating, updating, exporting, and removing environments, plus more

*Managing channels*
: Information about channels and how they are searched through when installing packages

*Managing packages*
: Details related to how to find, install, remove, and update packages in a given environment

*Managing python*
: Supported versions of Python and tips for updating and using multiple Python versions

*Managing virtual packages*
: Learn what virtual packages are and conda uses them

*View command line help*
: Get help on the command line for any conda command

### Tutorials

*Creating custom channels*
: Tutorial walking you through how to create a custom channel and serve it from your local computer

*Creating projects with conda*
: Learn how to start a new project with conda using a `environment.yml` file to manage your dependencies

## 4.1.4 Configuration

### Using the .condarc conda configuration file

### Overview

The conda configuration file, `.condarc`, is an optional runtime configuration file that allows advanced users to configure various aspects of conda, such as which channels it searches for packages, proxy settings, and environment directories. For all of the conda configuration options, see the *configuration page*.

**Note**

A `.condarc` file can also be used in an administrator-controlled installation to override the users' configuration. See *Administering a multi-user conda installation*.

The `.condarc` file can change many parameters, including:

- Where conda looks for packages.
- If and how conda uses a proxy server.
- Where conda lists known environments.
- Whether to update the Bash prompt with the currently activated environment name.
- Whether user-built packages should be uploaded to Anaconda.org.

- What default packages or features to include in new environments.

### Creating and editing

The `.condarc` file is not included by default, but it is automatically created in your home directory the first time you run the `conda config` command. To create or modify a `.condarc` file, open a terminal and enter the `conda config` command.

The `.condarc` configuration file follows simple YAML syntax.

**Example:**

```
conda config --add channels conda-forge
```

Alternatively, you can open a text editor such as Notepad on Windows, TextEdit on macOS, or VS Code. Name the new file `.condarc` and save it to your user home directory or root directory. To edit the `.condarc` file, open it from your home or root directory and make edits in the same way you would with any other text file. If the `.condarc` file is in the root environment, it will override any in the home directory.

You can find information about your `.condarc` file by typing `conda info` in your terminal. This will give you information about your `.condarc` file, including where it is located.

You can also download a *sample .condarc file* to edit in your editor and save to your user home directory or root directory.

To set configuration options, edit the `.condarc` file directly or use the `conda config --set` command.

**Example:**

To set the `auto_update_conda option` to `False`, run:

```
conda config --set auto_update_conda False
```

For a complete list of conda config commands, see the *command reference*. The same list is available at the terminal by running `conda config --help`. You can also see the conda channel configuration for more information.

Conda supports a wide range of configuration options. This page gives a non-exhaustive list of the most frequently used options and their usage. For a complete list of all available options for your version of conda, use the `conda config --describe` command.

### Searching for .condarc

Conda looks in the following locations for a `.condarc` file:

```
if on_win:
    SEARCH_PATH = (
        "C:/ProgramData/conda/.condarc",
        "C:/ProgramData/conda/condarc",
        "C:/ProgramData/conda/condarc.d",
    )
else:
    SEARCH_PATH = (
        "/etc/conda/.condarc",
        "/etc/conda/condarc",
        "/etc/conda/condarc.d/",
        "/var/lib/conda/.condarc",
```

(continues on next page)

(continued from previous page)

```
        "/var/lib/conda/condarc",
        "/var/lib/conda/condarc.d/",
    )

SEARCH_PATH += (
    "$CONDA_ROOT/.condarc",
    "$CONDA_ROOT/condarc",
    "$CONDA_ROOT/condarc.d/",
    "$XDG_CONFIG_HOME/conda/.condarc",
    "$XDG_CONFIG_HOME/conda/condarc",
    "$XDG_CONFIG_HOME/conda/condarc.d/",
    "~/.config/conda/.condarc",
    "~/.config/conda/condarc",
    "~/.config/conda/condarc.d/",
    "~/.conda/.condarc",
    "~/.conda/condarc",
    "~/.conda/condarc.d/",
    "~/.condarc",
    "$CONDA_PREFIX/.condarc",
    "$CONDA_PREFIX/condarc",
    "$CONDA_PREFIX/condarc.d/",
    "$CONDARC",
)
```

`XDG_CONFIG_HOME` is the path to where user-specific configuration files should be stored defined following The XDG Base Directory Specification (XDGBDS). Default to $HOME/.config should be used. `CONDA_ROOT` is the path for your base conda install. `CONDA_PREFIX` is the path to the current active environment. `CONDARC` must be a path to a file named `.condarc`, `condarc`, or end with a YAML suffix (`.yml` or `.yaml`).

**Note**

Any condarc files that exist in any of these special search path directories need to end in a valid yaml extension (".yml" or ".yaml").

### Conflict merging strategy

When conflicts between configurations arise, the following strategies are employed:

- Lists - merge
- Dictionaries - merge
- Primitive - clobber

### Precedence

The precedence by which the conda configuration is built out is shown below. Each new arrow takes precedence over the ones before it. For example, config files (by parse order) will be superseded by any of the other configuration options. Configuration environment variables (formatted like `CONDA_<CONFIG NAME>`) will always take precedence over the other 3.


### Obtaining information from the .condarc file

You can use the following commands to get the effective settings for conda. The effective settings are those that have merged settings from all the sources mentioned above.

To get all keys and their values:

```
conda config --get
```

To get the value of a specific key, such as channels:

```
conda config --get channels
```

To show all the configuration file sources and their contents:

```
conda config --show-sources
```

### Saving settings to your .condarc file

The `.condarc` file can also be modified via conda commands. Below are several examples of how to do this.

To add a new value, such as http://conda.anaconda.org/mutirri, to a specific key, such as channels:

```
conda config --add channels http://conda.anaconda.org/mutirri
```

To remove an existing value, such as http://conda.anaconda.org/mutirri from a specific key, such as channels:

```
conda config --remove channels http://conda.anaconda.org/mutirri
```

To remove a key, such as channels, and all of its values:

```
conda config --remove-key channels
```

To configure channels and their priority for a single environment, make a `.condarc` file in the *root directory of that environment*.

### Sample .condarc file

Because the `.condarc` file is just a YAML file, it means that it can be edited directly. Below is an example `.condarc` file:

```
# This is a sample .condarc file.
# It adds the r Anaconda.org channel and enables
# the show_channel_urls option.

# channel locations. These override conda defaults, i.e., conda will
# search *only* the channels listed here, in the order given.
# Use "defaults" to automatically include all default channels.
# Non-url channels will be interpreted as Anaconda.org usernames
# (this can be changed by modifying the channel_alias key; see below).
# The default is just 'defaults'.
channels:
  - r
  - defaults

# Show channel URLs when displaying what is going to be downloaded
# and in 'conda list'. The default is False.
show_channel_urls: True

# For more information about this file see:
# https://conda.io/docs/user-guide/configuration/use-condarc.html
```

### Settings

This page contains an overview of many important settings available in conda with examples where possible.

#### General configuration

##### `channels`: Channel locations

Listing channel locations in the `.condarc` file overrides conda defaults, causing conda to search only the channels listed there in the order given.

Use `defaults` to automatically include all default channels. Non-URL channels are interpreted as Anaconda.org user or organization names. You can change this by modifying the `channel_alias` as described in *channel_alias: Set a channel alias*. The default is just `defaults`.

**Example:**

```
channels:
  - <anaconda_dot_org_username>
  - http://some.custom/channel
  - file:///some/local/directory
  - defaults
```

To select channels for a single environment, put a `.condarc` file in the root directory of that environment (or use the `--env` option when using `conda config`).

**Example:** If you have installed Miniconda with Python 3 in your home directory and the environment is named "flowers", the path may be:

```
~/miniconda3/envs/flowers/.condarc
```

### `default_channels`: Default channels

Normally, the defaults channel points to several channels at the repo.anaconda.com repository, but if `default_channels` is defined, it sets the new list of default channels. This is especially useful for airgapped and enterprise installations.

To ensure that all users only pull packages from an on-premises repository, an administrator can set both *channel alias* and `default_channels`.

```
default_channels:
  - http://some.custom/channel
  - file:///some/local/directory
```

### `channel_settings`: Extra settings for individual channels

Added in version 23.3.0.

With `channel_settings`, it is possible to add extra configuration options for individual channels. This is currently used to register additional authentication handlers for conda via the *Auth Handlers* plugin hook, but may also accommodate more use cases in the future.

Here is an example of how it may be defined provided there was an available authentication handler called, "test-auth-handler" registered via the aforementioned plugin hook:

```
channel_settings:
   - channel: https://some.custom/channel
     auth: test-auth-handler
     user: my-user-account
   - channel: https://some.base-url-prefix/*
     auth: another-auth-handler
```

**Note**

Each entry in `channel_settings` needs to define the `channel` attribute so that the configuration knows which channel these settings are associated with. The `channel` attribute may specify a glob-like URL pattern for matching. Note that in this case, the HTTP schema must match exactly to the channel URL, so a pattern like * is not valid.

### `allowlist_channels` and `denylist_channels`: Allow or deny specific channels

Added in version 24.9.0: The `denylist_channels` setting was introduced in conda 24.9.0 complementing the existing `allowlist_channels` setting.

With `allowlist_channels` and `denylist_channels`, you can allow or deny specific channels from being used in conda operations. This is useful for restricting the channels that conda can access, especially in enterprise or multi-user environments.

The denylist takes precedence over the allowlist. If a channel is in both lists, it is denied.

**Examples:**

An example which allows the `defaults` and `conda-forge` channels with the `allowlist_channels` setting is:

```
allowlist_channels:
  - defaults
  - conda-forge
```

An example which denies the `conda-forge` channel with the `denylist_channels` setting is:

```
denylist_channels:
  - conda-forge
```

An example which explicitly allows the `defaults` channel but denies the `conda-forge` channel by using both the `allowlist_channels` and `denylist_channels` settings is:

```
allowlist_channels:
  - defaults
denylist_channels:
  - conda-forge
```

An example to show that channels are automatically normalized based on their base URLs, so you can use either the full channel URL or just the base URL:

```
allowlist_channels:
  - defaults
denylist_channels:
  - https://conda.anaconda.org/conda-forge/linux-64
```

An example that denies using `defaults` (which maps to the *default_channels*) configuration option:

```
denylist_channels:
  - defaults
```

**Note**

The *defaults channel* points to a list of channels at the repo.anaconda.com repository by default.

An example to explicitly deny the channels that are hosted on `repo.anaconda.com`:

```
denylist_channels:
  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
  - https://repo.anaconda.com/pkgs/msys2
```

#### auto_update_conda: Update conda automatically

When `True`, conda updates itself any time a user updates or installs a package in the root environment. When `False`, conda updates itself only if the user manually issues a `conda update` command. The default is `True`.

**Example:**

```
auto_update_conda: False
```

#### always_yes: Always yes

Choose the `yes` option whenever asked to proceed, such as when installing. Same as using the `--yes` flag at the command line. The default is `False`.

**Example:**

```
always_yes: True
```

#### show_channel_urls: Show channel URLs

Show channel URLs in `conda list` and when displaying what is going to be downloaded. The default is `False`.

**Example:**

```
show_channel_urls: True
```

#### changeps1: Change command prompt

When using `conda activate`, change the command prompt from `$PS1` to include the activated environment. The default is `True`.

**Example:**

```
changeps1: False
```

#### add_pip_as_python_dependency: Add pip as Python dependency

Add pip, wheel, and setuptools as dependencies of Python. This ensures that pip, wheel, and setuptools are always installed any time Python is installed. The default is `True`.

**Example:**

```
add_pip_as_python_dependency: False
```

### use_pip: Use pip

Use pip when listing packages with `conda list`. This does not affect any conda command or functionality other than the output of the command `conda list`. The default is `True`.

**Example:**

```
use_pip: False
```

### proxy_servers: Configure conda for use behind a proxy server

By default, proxy settings are pulled from the HTTP_PROXY and HTTPS_PROXY environment variables or the system. Setting them here overrides that default:

```
proxy_servers:
    http: http://user:pass@corp.com:8080
    https: http://user:pass@corp.com:8080
```

**Mixing HTTPS and HTTP**

The protocol in the URL (either `http://` or `https://`) should match the actual protocol of your proxy server. The keys `http` and `https` in the above example merely indicate the type of traffic to route, not the protocol of the proxy server itself. Ensure that both keys use the correct protocol based on your proxy server's configuration.

To give a proxy for a specific scheme and host, use the `scheme://hostname` form for the key. This matches for any request to the given scheme and exact host name:

```
proxy_servers:
  'http://10.20.1.128': 'http://10.10.1.10:5323'
```

If you do not include the username and password or if authentication fails, conda prompts for a username and password.

If your password contains special characters, you need to escape them as described in Percent-encoding reserved characters on Wikipedia.

Be careful not to use `http` when you mean `https` or `https` when you mean `http`.

### ssl_verify: SSL verification

If you are behind a proxy that does SSL inspection, such as a Cisco IronPort Web Security Appliance (WSA), you may need to use `ssl_verify` to override the SSL verification settings.

By default, this variable is `True`, which means that SSL verification is used and conda verifies certificates for SSL connections. Setting this variable to `False` disables the connection's normal security and is not recommended:

```
ssl_verify: False
```

Added in version 23.9.0: The `ssl_verify: truststore` setting is only available with conda 23.9.0 or later and using Python 3.10 or later.

If the certificate authority is already trusted by the operating system, for instance because it was installed by a system administrator, you can tell conda to use the operating system certificate store by setting `ssl_verify` to "truststore":

```
ssl_verify: truststore
```

You can also set `ssl_verify` to a string path to a certificate, which can be used to verify SSL connections:

```
ssl_verify: corp.crt
```

#### `offline`: Offline mode only

Filters out all channel URLs that do not use the `file://` protocol. The default is `False`.

**Example:**

```
offline: True
```

### Advanced configuration

#### `allow_softlinks`: Allow soft-linking

When `allow_softlinks` is `True`, conda uses hard links when possible and soft links (symlinks) when hard links are not possible, such as when installing on a different file system than the one that the package cache is on.

When `allow_softlinks` is `False`, conda still uses hard links when possible, but when it is not possible, conda copies files. Individual packages can override this option, specifying that certain files should never be soft linked.

The default is `False`.

**Example:**

```
allow_softlinks: False
```

#### `channel_alias`: Set a channel alias

Whenever you use the `-c` or `--channel` flag to give conda a channel name that is not a URL, conda prepends the `channel_alias` to the name that it was given. The default `channel_alias` is https://conda.anaconda.org.

If `channel_alias` is set to `https://my.anaconda.repo:8080/conda/`, then a user who runs the command `conda install -c conda-forge some-package` will install the package some-package from `https://my.anaconda.repo:8080/conda/conda-forge`.

For example, the command:

```
conda install --channel asmeurer <package>
```

is the same as:

```
conda install --channel https://conda.anaconda.org/asmeurer <package>
```

You can set `channel_alias` to your own repository.

**Example:** To set `channel_alias` to your repository at https://your.repo.com:

```
channel_alias: https://your.repo/
```

On Windows, you must include a slash ("/") at the end of the URL:

**Example:** https://your.repo/conda/

When `channel_alias` set to your repository at https://your.repo.com:

```
conda install --channel jsmith <package>
```

is the same as:

```
conda install --channel https://your.repo.com/jsmith <package>
```

### `create_default_packages`: Always add packages by default

When creating new environments, add the specified packages by default. The default packages are installed in every environment you create. You can override this option at the command prompt with the `--no-default-packages` flag. The default is to not include any packages.

**Example:**

```
create_default_packages:
  - pip
  - ipython
  - scipy=0.15.0
```

### `track_features`: Track features

Enable certain features to be tracked by default. The default is to not track any features. This is similar to adding MKL to the `create_default_packages` list.

**Example:**

```
track_features:
  - mkl
```

### `update_dependencies`: Disable updating of dependencies

By default, `conda install` updates the given package to the latest version and installs any dependencies necessary for that package. However, if dependencies that satisfy the package's requirements are already installed, conda will not update those packages to the latest version.

In this case, if you would prefer that conda update all dependencies to the latest version that is compatible with the environment, set `update_dependencies` to `True`.

The default is `False`.

**Example:**

```
update_dependencies: True
```

> **Note**
>
> Conda still ensures that dependency specifications are satisfied. Thus, some dependencies may still be updated or, conversely, this may prevent packages given at the command line from being updated to their latest versions. You can always specify versions at the command line to force conda to install a given version, such as `conda install numpy=1.9.3`.

To avoid updating only specific packages in an environment, a better option may be to pin them. For more information, see *Preventing packages from updating (pinning)*.

### `disallow`: Disallow installation of specific packages

Disallow the installation of certain packages. The default is to allow installation of all packages.

**Example:**

```
disallow:
  - anaconda
```

### `add_anaconda_token`: Add Anaconda.org token to automatically see private packages

When the channel alias is Anaconda.org or an Anaconda Server GUI, you can set the system configuration so that users automatically see private packages. Anaconda.org was formerly known as binstar.org. This uses the Anaconda command-line client, which you can install with `conda install anaconda-client`, to automatically add the token to the channel URLs.

The default is `True`.

**Example:**

```
add_anaconda_token: False
```

> **Note**
>
> Even when set to `True`, this setting is enabled only if the Anaconda command-line client is installed and you are logged in with the `anaconda login` command.

### `envs_dirs`: Specify environment directories

Specify directories in which environments are located. If this key is set, the root prefix `envs_dir` is not used unless explicitly included. This key also determines where the package caches are located.

For each envs here, `envs/pkgs` is used as the pkgs cache, except for the standard `envs` directory in the root directory, for which the normal `root_dir/pkgs` is used.

**Example:**

```
envs_dirs:
  - ~/my-envs
  - /opt/anaconda/envs
```

The CONDA_ENVS_PATH environment variable overwrites the envs_dirs setting:

- For macOS and Linux: CONDA_ENVS_PATH=~/my-envs:/opt/anaconda/envs
- For Windows: set CONDA_ENVS_PATH=C:\Users\joe\envs;C:\Anaconda\envs

### pkgs_dirs: Specify package directories

Specify directories in which packages are located. If this key is set, the root prefix pkgs_dirs is not used unless explicitly included.

If the pkgs_dirs key is not set, then envs/pkgs is used as the pkgs cache, except for the standard envs directory in the root directory, for which the normal root_dir/pkgs is used.

**Example:**

```
pkgs_dirs:
  - /opt/anaconda/pkgs
```

The CONDA_PKGS_DIRS environment variable overwrites the pkgs_dirs setting:

- For macOS and Linux: CONDA_PKGS_DIRS=/opt/anaconda/pkgs
- For Windows: set CONDA_PKGS_DIRS=C:\Anaconda\pkgs

### use_only_tar_bz2: Force conda to download only .tar.bz2 packages

Conda 4.7 introduced a new .conda package file format. .conda is a more compact and faster alternative to .tar.bz2 packages. It's thus the preferred file format to use where available.

Nevertheless, it's possible to force conda to only download .tar.bz2 packages by setting the use_only_tar_bz2 boolean to True.

The default is False.

**Example:**

```
use_only_tar_bz2: True
```

> **Note**
>
> This is forced to True if conda-build is installed and older than 3.18.3, because older versions of conda break when conda feeds it the new file format.

### console: Configure display type

Added in version 24.11.0: the console setting is only available after this version.

The console setting allows you to modify the way output is rendered for conda commands. This setting is primarily used as a way to select new reporter backends made available by plugins.

For example, a plugin may create a new reporter backend called "colors". As a user, you would configure it in your .condarc file as shown below:

```
console: colors
```

or specify it on the command line with the `--console` option

```
conda info --console=colors
```

### Conda-build configuration

#### `root-dir`: Specify conda-build output root directory

Build output root directory. You can also set this with the `CONDA_BLD_PATH` environment variable. The default is `<CONDA_PREFIX>/conda-bld/`. If you do not have write permissions to `<CONDA_PREFIX>/conda-bld/`, the default is `~/conda-bld/`.

**Example:**

```
conda-build:
    root-dir: ~/conda-builds
```

#### `output_folder`: Specify conda-build build folder (conda-build 3.16.3+)

Folder to dump output package to. Packages are moved here if build or test succeeds. If unset, the output folder corresponds to the same directory as `root-dir`: the root build directory. .. code-block:: yaml

**conda-build:**
output_folder: conda-bld

#### `pkg_version`: Specify conda-build package version

Conda package version to create. Use 2 for `.conda` packages. If not set, conda-build defaults to `.tar.bz2`.

```
conda-build:
   pkg_format: 2
```

#### `anaconda_upload`: Automatically upload conda-build packages to Anaconda.org

Automatically upload packages built with conda-build to Anaconda.org. The default is `False`.

**Example:**

```
anaconda_upload: True
```

#### anaconda_token: Token to be used for Anaconda.org uploads (conda-build 3.0+)

Tokens are a means of authenticating with Anaconda.org without logging in. You can pass your token to conda-build with this `.condarc` setting, or with a CLI argument. This is unset by default. Setting it implicitly enables `anaconda_upload`.

```
conda-build:
    anaconda_token: gobbledygook
```

#### quiet: Limit build output verbosity (conda-build 3.0+)

Conda-build's output verbosity can be reduced with the `quiet` setting. For more verbosity, use the CLI flag `--debug`.

```
conda-build:
    quiet: true
```

#### filename_hashing: Disable filename hashing (conda-build 3.0+)

Conda-build 3 adds hashes to filenames to allow greater customization of dependency versions. If you find this disruptive, you can disable the hashing with the following config entry:

```
conda-build:
    filename_hashing: false
```

> **Warning**
>
> Conda-build does not check when clobbering packages. If you utilize conda-build 3's build matrices with a build configuration that is not reflected in the build string, packages will be missing due to clobbering.

#### no_verify: Disable recipe and package verification (conda-build 3.0+)

By default, conda-build uses conda-verify to ensure that your recipe and package meet some minimum sanity checks. You can disable these:

```
conda-build:
    no_verify: true
```

#### set_build_id: Disable per-build folder creation (conda-build 3.0+)

By default, conda-build creates a new folder for each build, named for the package name plus a timestamp. This allows you to do multiple builds at once. If you have issues with long paths, you may need to disable this behavior. You should first try to change the build output root directory with the `root-dir` setting described above, but fall back to this as necessary:

```
conda-build:
    set_build_id: false
```

### `skip_existing`: Skip building packages that already exist (conda-build 3.0+)

By default, conda-build builds all recipes that you specify. You can instead skip recipes that are already built. A recipe is skipped if and only if *all* of its outputs are available on your currently configured channels.

```
conda-build:
    skip_existing: true
```

### `include_recipe`: Omit recipe from package (conda-build 3.0+)

By default, conda-build includes the recipe that was used to build the package. If this contains sensitive or proprietary information, you can omit the recipe.

```
conda-build:
    include_recipe: false
```

**Note**

If you do not include the recipe, you cannot use conda-build to test the package after the build completes. This means that you cannot split your build and test steps across two distinct CLI commands (`conda build --notest recipe` and `conda build -t recipe`). If you need to omit the recipe and split your steps, your only option is to remove the recipe files from the tarball artifacts after your test step. Conda-build does not provide tools for doing that.

### `activate`: Disable activation of environments during build/test (conda-build 3.0+)

By default, conda-build activates the build and test environments prior to executing the build or test scripts. This adds necessary PATH entries, and also runs any activate.d scripts you may have. If you disable activation, the PATH will still be modified, but the activate.d scripts will not run. This is not recommended, but some people prefer this.

```
conda-build:
    activate: false
```

### `long_test_prefix`: Disable long prefix during test (conda-build 3.16.3+)

By default, conda-build uses a long prefix for the test prefix. If you have recipes that fail in long prefixes but would still like to test them in short prefixes, you can disable the long test prefix. This is not recommended.

```
conda-build:
    long_test_prefix: false
```

The default is `true`.

### `pypirc`: PyPI upload settings (conda-build 3.0+)

Unset by default. If you have wheel outputs in your recipe, conda-build will try to upload them to the PyPI repository specified by the `pypi_repository` setting using credentials from this file path.

```
conda-build:
    pypirc: ~/.pypirc
```

### `pypi_repository`: PyPI repository to upload to (conda-build 3.0+)

Unset by default. If you have wheel outputs in your recipe, conda-build will try to upload them to this PyPI repository using credentials from the file specified by the `pypirc` setting.

```
conda-build:
    pypi_repository: pypi
```

### Expansion of environment variables

Conda expands environment variables in a subset of configuration settings. These are:

- `channel`
- `channel_alias`
- `channels`
- `client_cert_key`
- `client_cert`
- `custom_channels`
- `custom_multichannels`
- `default_channels`
- `envs_dirs`
- `envs_path`
- `migrated_custom_channels`
- `pkgs_dirs`
- `proxy_servers`
- `verify_ssl`
- `allowlist_channels`
- `denylist_channels`

This allows you to store the credentials of a private repository in an environment variable, like so:

```
channels:
  - https://${USERNAME}:${PASSWORD}@my.private.conda.channel
```

### Configuring number of threads

You can use your `.condarc` file or environment variables to add configuration to control the number of threads. You may want to do this to tweak conda to better utilize your system. If you have a very fast SSD, you might increase the number of threads to shorten the time it takes for conda to create environments and install/remove packages.

#### repodata_threads

- Default number of threads: None
- Threads used when downloading, parsing, and creating repodata structures from `repodata.json` files. Multiple downloads from different channels may occur simultaneously. This speeds up the time it takes to start solving.

#### verify_threads

- Default number of threads: 1
- Threads used when verifying the integrity of packages and files to be installed in your environment. Defaults to 1, as using multiple threads here can run into problems with slower hard drives.

#### execute_threads

- Default number of threads: 1
- Threads used to unlink, remove, link, or copy files into your environment. Defaults to 1, as using multiple threads here can run into problems with slower hard drives.

#### default_threads

- Default number of threads: None
- When set, this value is used for all of the above thread settings. With its default setting (None), it does not affect the other settings.

Setting any of the above can be done in `.condarc` or with conda config:

At your terminal:

```
conda config --set repodata_threads 2
```

In `.condarc`:

```
verify_threads: 4
```

### Administering a multi-user conda installation

By default, conda and all of the packages it installs are installed locally with a user-specific configuration. Administrative privileges are not required, and no upstream files or other users are affected by the installation.

You can make conda and any number of packages available to a group of one or more users, while preventing these users from installing unwanted packages with conda:

1. Install conda and the allowed packages, if any, in a location that is under administrator control and accessible to users.
2. Create a *.condarc system configuration file* in the root directory of the installation. This system-level configuration file will override any user-level configuration files installed by the user.

Each user accesses the central conda installation, which reads settings from the user `.condarc` configuration file located in their home directory. The path to the user file is the same as the root environment prefix displayed by `conda info`, as shown in *User configuration file* below. The user `.condarc` file is limited by the system `.condarc` file.

System configuration settings are commonly used in a system `.condarc` file but may also be used in a user `.condarc` file. All user configuration settings may also be used in a system `.condarc` file.

For information about settings in the `.condarc` file, see *Using the .condarc conda configuration file*.

### Example administrator-controlled installation

The following example describes how to view the system configuration file, review the settings, compare it to a user's configuration file, and determine what happens when the user attempts to access a file from a blocked channel. It then describes how the user must modify their configuration file to access the channels allowed by the administrator.

### System configuration file

1. The system configuration file must be in the top-level conda installation directory. Check the path where `conda` is located, e.g. in a miniconda installation

```
$ which conda
/tmp/miniconda/bin/conda
```

2. View the contents of the `.condarc` file in the administrator's directory:

```
cat /tmp/miniconda/.condarc
```

   The following administrative `.condarc` file uses the `#!final` flag to specify the channels, default channels, and channel_alias available to the user.

```
$ cat /tmp/miniconda/.condarc

channels:                                    #!final
  - admin

channel_alias: https://conda.anaconda.org/  #!final
```

The `#!final` flag is very similar to the `!important` rule in CSS; any parameter within the `.condarc` that is trailed by the `#!final` cannot be overwritten by any other `.condarc` source. For more information on this flag, see the Anaconda Blog on the subject.

Because the `#!final` flag has been used and the channel defaults are not explicitly specified, users are disallowed from downloading packages from the default channels. You can check this in the next procedure.

### User configuration file

1. Check the location of the user's conda installation:

```
$ conda info
Current conda install:
. . .
       channel URLs : https://repo.anaconda.com/pkgs/free/osx-64/
                      https://repo.anaconda.com/pkgs/pro/osx-64/
       config file : /Users/username/.condarc
```

   The `conda info` command shows that conda is using the user's `.condarc` file, located at `/Users/username/.condarc` and that the default channels such as `repo.anaconda.com` are listed as channel URLs.

2. View the contents of the administrative `.condarc` file in the directory that was located in step 1:

```
$ cat ~/.condarc
channels:
  - defaults
```

   This user's `.condarc` file specifies only the default channels, but the administrator config file has blocked default channels by specifying that only `admin` is allowed. If this user attempts to search for a package in the default channels, they get a message telling them what channels are allowed:

```
$ conda search flask
Fetching package metadata:
Error: URL 'http://repo.anaconda.com/pkgs/pro/osx-64/' not
in allowed channels.
Allowed channels are:
 - https://conda.anaconda.org/admin/osx-64/
```

   This error message tells the user to add the `admin` channel to their configuration file.

3. The user must edit their local `.condarc` configuration file to access the package through the admin channel:

```
channels:
  - admin
```

   The user can now search for packages in the allowed `admin` channel.

### Mirroring channels

The conda configuration system has several keys that can be used to set up a mirrored context.

### The default setup

By default, conda can serve packages from two main locations:

- repo.anaconda.com: this is where defaults points to by default. The default value has been marked for pending deprecation as of conda 24.9.0, but it is still the default for older versions. In conda 25.3.0, this default will be removed.

  **This base location is hardcoded in the default value of default_channels:**

  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
  - https://repo.anaconda.com/pkgs/msys2

- conda.anaconda.org: this is where conda clients look up community channels like conda-forge or bioconda. This base location can be configured via channel_alias.

So, when it comes to mirroring these channels, you have to account for those two locations.

### Mirror defaults

Use default_channels to overwrite the *default configuration*. For example:

```
default_channels:
    - https://my-mirror.com/pkgs/main
    - https://my-mirror.com/pkgs/r
    - https://my-mirror.com/pkgs/msys2
```

### Mirror all community channels

Redefine channel_alias to point to your mirror. For example:

```
channel_alias: https://my-mirror.com
```

This will make conda look for all community channels at https://my-mirror.com/conda-forge, https://my-mirror.com/bioconda, etc.

### Mirror only some community channels

If you want to mirror only some community channels, you must use custom_channels. This takes precedence over channel_alias. For example:

```
custom_channels:
    conda-forge: https://my-mirror.com/conda-forge
```

With this configuration, conda-forge will be looked up at https://my-mirror.com/conda-forge. All other community channels will be looked up at https://conda.anaconda.org.

**Note**

Feel free to explore all the available options in *Configuration*.

### Disabling SSL verification

Using conda with SSL is strongly recommended, but it is possible to disable SSL and it may be necessary to disable SSL in certain cases.

Some corporate environments use proxy services that use Man-In-The-Middle (MITM) attacks to sniff encrypted traffic. These services can interfere with SSL connections such as those used by conda and pip to download packages from repositories such as PyPI.

If you encounter this interference, you should set up the proxy service's certificates so that the `requests` package used by conda can recognize and use the certificates.

For cases where this is not possible, conda-build versions 3.0.31 and higher have an option that disables SSL certificate verification and allows this traffic to continue.

`conda skeleton pypi` can disable SSL verification when pulling packages from a PyPI server over HTTPS.

> **Warning**
>
> This option causes your computer to download and execute arbitrary code over a connection that it cannot verify as secure. This is not recommended and should only be used if necessary. Use this option at your own risk.

To disable SSL verification when using `conda skeleton pypi`, set the `SSL_NO_VERIFY` environment variable to either `1` or `True` (case insensitive).

On *nix systems:

```
SSL_NO_VERIFY=1 conda skeleton pypi a_package
```

And on Windows systems:

```
set SSL_NO_VERIFY=1
conda skeleton pypi a_package
set SSL_NO_VERIFY=
```

We recommend that you unset this environment variable immediately after use. If it is not unset, some other tools may recognize it and incorrectly use unverified SSL connections.

Using this option will cause `requests` to emit warnings to STDERR about insecure settings. If you know that what you're doing is safe, or have been advised by your IT department that what you're doing is safe, you may ignore these warnings.

### Disabling SSL verification via conda settings

In addition to disabling SSL via environment variables, you can disable it by setting *ssl_verify* to *false* in your config files. To do so, run the following commands to disable and enable it:

```
conda config --set ssl_verify False
# Run conda commands with SSL disabled
conda config --set ssl_verify True
```

### Using non-standard certificates

Using conda behind a firewall may require using a non-standard set of certificates, which requires custom settings.

If you are using a non-standard set of certificates, then the requests package requires the setting of `REQUESTS_CA_BUNDLE`. If you receive an error with self-signed certifications, you may consider unsetting `REQUESTS_CA_BUNDLE` as well as `CURL_CA_BUNDLE` and disabling SSL verification to create a conda environment over HTTP.

You may need to set the conda environment to use the root certificate provided by your company rather than conda's generic ones.

One workflow to resolve this on macOS is:

- Open Chrome, got to any website, click on the lock icon on the left of the URL. Click on «Certificate» on the dropdown. In the next window you see a stack of certificates. The uppermost (aka top line in window) is the root certificate (e.g. Zscaler Root CA).
- Open macOS keychain, click on «Certificates» and choose among the many certificates the root certificate that you just identified. Export this to any folder of your choosing.
- Convert this certificate with OpenSSL: `openssl x509 -inform der -in /path/to/your/certificate.cer -out /path/to/converted/certificate.pem`
- For a quick check, set your shell to acknowledge the certificate: `export REQUESTS_CA_BUNDLE=/path/to/converted/certificate.pem`
- To set this permanently, open your shell profile (e.g. `.bashrc` or `.zshrc`) and add this line: `export REQUESTS_CA_BUNDLE=/path/to/converted/certificate.pem`. Now exit your terminal/shell and reopen. Check again.

### Using Custom Locations for Environment and Package Cache

For any given conda installation, the two largest folders in terms of disk space are often the `envs` and `pkgs` folders that store created environments and downloaded packages, respectively. If the location where conda is installed has limited disk space and another location with more disk space is available on the same computer, we can change where conda saves its environments and packages with the settings `envs_dirs` and `pkgs_dirs`, respectively.

Assuming conda is installed in the user's home directory and the the folder `/nfs/volume/user` with more disk space is writable, the best way to configure this is by adding the following entries to the `.condarc` file in the user's home directory:

```
envs_dirs:
  - /nfs/volume/user/conda_envs
pkgs_dirs:
  - /nfs/volume/user/conda_pkgs
```

In the example above, we tell conda to use the folder `/nfs/volume/user/conda_envs` to store all of the environments we create, and we tell conda to use `/nfs/volume/user/conda_pkgs` to store all of the packages that we download.

To save even more space, the contents of `/nfs/volume/user/conda_pkgs` will be hard linked to the environments in `/nfs/volume/user/conda_envs` when possible. This means that `pkgs_dirs` will normally take up the most space for a conda installation. But, when hard linking is not possible, the files will be copied over to the environment which means each new environment increases the amount of disk space taken. To ensure this hard linking works properly, we recommend to always store the `envs_dirs` and `pkgs_dirs` on the same mounted volume.

### Improving interoperability with pip

The conda 4.6.0 release added improved support for interoperability between conda and pip. This feature is still experimental and is therefore off by default.

With this interoperability, conda can use pip-installed packages to satisfy dependencies, cleanly remove pip-installed software, and replace them with conda packages when appropriate.

If you'd like to try the feature, you can set this `.condarc` setting:

```
conda config --set prefix_data_interoperability True
```

> **Note**
>
> Setting `prefix_data_interoperability` to `True` may slow down conda.

Even without activating this feature, conda now understands pip metadata more intelligently. For example, if we create an environment with conda:

```
conda create -y -n some_pip_test python=3.7 imagesize=1.0
```

Then we update imagesize in that environment using pip:

```
conda activate some_pip_test
pip install -U imagesize
```

Prior to conda 4.6.0, the `conda list` command returned ambiguous results:

```
imagesize                 1.1.0

imagesize                 1.0.0 py37_0
```

Conda 4.6.0 now shows only one entry for imagesize (the newer pip entry):

```
imagesize                 1.1.0 pypi_0    pypi
```

### Using the free channel

The free channel contains packages created prior to September 26, 2017. Prior to conda 4.7, the free channel was part of the `defaults` channel. Read more about the *defaults channel*.

Removing the `free` channel reduced conda's search space and hid old software. That old software could have incompatible constraint information. Read more about why we made this change.

If you still need the content from the `free` channel to reproduce old environments, you can re-add the channel following the directions below.

Changed in version 24.9.0: The `restore_free_channel` option has been marked for pending deprecation with removal in conda 25.9.0.

To achieve the same effect, you may add the `free` channel to your the *defaults channel* using the regular `condarc` configuration.

On UNIX-style systems:

```
default_channels:
    - https://repo.anaconda.com/pkgs/main
    - https://repo.anaconda.com/pkgs/free
    - https://repo.anaconda.com/pkgs/r
```

On Windows:

```
default_channels:
    - https://repo.anaconda.com/pkgs/main
    - https://repo.anaconda.com/pkgs/free
    - https://repo.anaconda.com/pkgs/r
    - https://repo.anaconda.com/pkgs/msys2
```

Note that the free channel is listed after the main channel.

### Adding the free channel to defaults

If you want to add the `free` channel back into your default list, use the command:

```
conda config --set restore_free_channel true
```

The order of the channels is important. Using the above command will restore the `free` channel in the correct order.

### Changing .condarc

You can also add the `free` channel back into your defaults by changing the `.condarc` file itself.

Add the following to the conda section of your `.condarc` file:

```
restore_free_channel: true
```

Read more about *Using the .condarc conda configuration file*.

### Package name changes

Some packages that are available in the `free` channel have different names in the `main` channel.

| Package name in `free` | Package name in `main` |
|---|---|
| dateutil | python-dateutil |
| gcc | gcc_linux-64 and similar |
| pil | pillow |
| ipython-notebook | now installable via notebook, a metapackage could be created |
| Ipython-qtconsole | now installable via qtconsole, a metapackage could be created |
| beautiful-soup | beautifulsoup4 |
| pydot-ng | pydot |

### Troubleshooting

You may encounter some errors, such as UnsatisfiableError or a PackagesNotFoundError.

An example of this error is:

```
$ conda create -n test -c file:///Users/jsmith/anaconda/conda-bld bad_pkg
Collecting package metadata: done
Solving environment: failed

UnsatisfiableError: The following specifications were found to be in conflict:
  - cryptography=2.6.1 -> openssl[version='>=1.1.1b,<1.1.2a']
  - python=3.7.0 -> openssl[version='>=1.0.2o,<1.0.3a']
Use "conda search <package> --info" to see the dependencies for each package.
```

This can occur if:

- you're trying to install a package that is only available in `free` and not in `main`.
- you have older environments in files you want to recreate. If those spec files reference packages that are in `free`, they will not show up.
- a package is dependent upon files found only in the free channel. Conda will not let you install the package if it cannot install the dependency, which the package requires to work.

If you encounter these errors, consider using a newer package than the one in `free`. If you want those older versions, you can *add the free channel back into your defaults*.

The following pages have information on how conda can be customized further through configuration.

***Using the .condarc conda configuration file***
: Learn how to use a settings file (`.condarc`) to override defaults and maintain settings across shell sessions

***Settings***
: View a list and definition of all the configuration settings that can be used within conda

***Administering a multi-user conda installation***
: How to set up conda as a system administrator for use by multiple users

***Mirroring channels***
: Explore how to configure your own channel server mirror with conda

***Disable SSL Verification***
: Disabling SSL may be necessary in very limited circumstances; learn how here

***Using non-standard certificates***
: Install and configure non-standard certifications for use with conda

***Using Custom Locations for Environment and Package Cache***
: How to configure different locations for storing environments and package cache

***Pip interoperability (experimental)***
: An experimental feature that makes conda operate better with pip (no longer supported)

***Free channel (deprecated)***
: Explanation of our deprecation of the free channel and how to restore it

### 4.1.5 Concepts

#### Commands

The `conda` command is the primary interface for managing installations of various packages. It can:

- Query and search the Anaconda package index and current Anaconda installation.
- Create new conda environments.
- Install and update packages into existing conda environments.

> **Tip**
>
> You can abbreviate many frequently used command options that are preceded by 2 dashes (`--`) to just 1 dash and the first letter of the option. So `--name` and `--envs` can be written as `-n` and `-e` respectively.

For full usage of each command, including abbreviations, see *commands*. You can see the same information at the command line by *viewing the command-line help*.

#### Packages

#### What is a package?

A package is a compressed tarball file (`.tar.bz2`) or `.conda` file that contains:

- system-level libraries.
- Python or other modules.
- executable programs and other components.
- metadata under the `info/` directory.
- a collection of files that are installed directly into an `install` prefix.

Conda keeps track of the dependencies between packages and platforms. The conda package format is identical across platforms and operating systems.

Only files, including symbolic links, are part of a conda package. Directories are not included. Directories are created and removed as needed, but you cannot create an empty directory from the tar archive directly.

#### .conda file format

The .conda file format was introduced in conda 4.7 as a more compact, and thus faster, alternative to a tarball.

The .conda file format consists of an outer, uncompressed ZIP-format container, with 2 inner compressed .tar files.

For the .conda format's initial internal compression format support, we chose Zstandard (zstd). The actual compression format used does not matter, as long as the format is supported by libarchive. The compression format may change in the future as more advanced compression algorithms are developed and no change to the .conda format is necessary. Only an updated libarchive would be required to add a new compression format to .conda files.

These compressed files can be significantly smaller than their bzip2 equivalents. In addition, they decompress much more quickly. .conda is the preferred file format to use where available, although we continue to provide .tar.bz2 files in tandem.

Read more about the introduction of the .conda file format.

**Note**

In conda 4.7 and later, you cannot use package names that end in ".conda" as they conflict with the .conda file format for packages.

### Using packages

- You may search for packages

```
conda search scipy
```

- You may install a package

```
conda install scipy
```

- You may build a package after installing conda-build

```
conda build my_fun_package
```

### Package structure

```
.
├── bin
│   └── pyflakes
├── info
│   ├── LICENSE.txt
│   ├── files
│   ├── index.json
│   ├── paths.json
│   └── recipe
└── lib
    └── python3.5
```

- bin contains relevant binaries for the package.
- lib contains the relevant library files (eg. the .py files).
- info contains package metadata.

### Metapackages

When a conda package is used for metadata alone and does not contain any files, it is referred to as a metapackage. The metapackage may contain dependencies to several core, low-level libraries and can contain links to software files that are automatically downloaded when executed. Metapackages are used to capture metadata and make complicated package specifications simpler.

An example of a metapackage is "anaconda," which collects together all the packages in the Anaconda Distribution installer. The command `conda create -n envname anaconda` creates an environment that exactly matches what would be created from the Anaconda Distribution installer. You can create metapackages with the `conda metapackage` command. Include the name and version in the command.

### Anaconda metapackage

The Anaconda metapackage is used in the creation of the Anaconda Distribution installers so that they have a set of packages associated with them. Each installer release has a version number, which corresponds to a particular collection of packages at specific versions. That collection of packages at specific versions is encapsulated in the Anaconda metapackage.

The Anaconda metapackage contains several core, low-level libraries, including compression, encryption, linear algebra, and some GUI libraries.

Read more about the Anaconda metapackage and Anaconda Distribution.

### Mutex metapackages

A mutex metapackage is a very simple package that has a name. It need not have any dependencies or build steps. Mutex metapackages are frequently an "output" in a recipe that builds some variant of another package. Mutex metapackages function as a tool to help achieve mutual exclusivity among packages with different names.

Let's look at some examples for how to use mutex metapackages to build NumPy against different BLAS implementations.

### Building NumPy with BLAS variants

If you build NumPy with MKL, you also need to build SciPy, scikit-learn, and anything else using BLAS also with MKL. It is important to ensure that these "variants" (packages built with a particular set of options) are installed together and never with an alternate BLAS implementation. This is to avoid crashes, slowness, or numerical problems. Lining up these libraries is both a build-time and an install-time concern. We'll show how to use metapackages to achieve this need.

Let's start with the metapackage `blas=1.0=mkl`: https://github.com/AnacondaRecipes/intel_repack-feedstock/blob/e699b12/recipe/meta.yaml#L108-L112

Note that `mkl` is a string of `blas`.

That metapackage is automatically added as a dependency using `run_exports` when someone uses the mkl-devel package as a build-time dependency: https://github.com/AnacondaRecipes/intel_repack-feedstock/blob/e699b12/recipe/meta.yaml#L124

By the same token, here's the metapackage for OpenBLAS: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L127-L131

And the `run_exports` for OpenBLAS, as part of openblas-devel: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L100

Fundamentally, conda's model of mutual exclusivity relies on the package name. OpenBLAS and MKL are obviously not the same package name, and thus are not mutually exclusive. There's nothing stopping conda from installing both at once. There's nothing stopping conda from installing NumPy with MKL and SciPy with OpenBLAS. The metapackage is what allows us to achieve the mutual exclusivity. It unifies the options on a single package name, but with a different build string. Automating the addition of the metapackage with `run_exports` helps ensure the library consumers (package builders who depend on libraries) will have correct dependency information to achieve the unified runtime library collection.

### Installing NumPy with BLAS variants

To specify which variant of NumPy that you want, you could potentially specify the BLAS library you want:

```
conda install numpy mkl
```

However, that doesn't actually preclude OpenBLAS from being chosen. Neither MKL nor its dependencies are mutually exclusive (meaning they do not have similar names and different version/build-string).

This pathway may lead to some ambiguity and solutions with mixed BLAS, so using the metapackage is recommended. To specify MKL-powered NumPy in a non-ambiguous way, you can specify the mutex package (either directly or indirectly):

```
conda install numpy "blas=*=mkl"
```

There is a simpler way to address this, however. For example, you may want to try another package that has the desired mutex package as a dependency.

OpenBLAS has this with its "nomkl" package: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L133-L147

Nothing should use "nomkl" as a dependency. It is strictly a utility for users to facilitate switching from MKL (which is the default) to OpenBLAS.

How did MKL become the default? The solver needs a way to prioritize some packages over others. We achieve that with an older conda feature called track_features that originally served a different purpose.

### Track_features

One of conda's optimization goals is to minimize the number of track_features needed to specify the desired specs. By adding track_features to one or more of the options, conda will de-prioritize it or "weigh it down." The lowest priority package is the one that would cause the most track_features to be activated in the environment. The default package among many variants is the one that would cause the least track_features to be activated.

There is a catch, though: any track_features must be unique. No two packages can provide the same track_feature. For this reason, our standard practice is to attach track_features to the metapackage associated with what we want to be non-default.

Take another look at the OpenBLAS recipe: https://github.com/AnacondaRecipes/openblas-feedstock/blob/ae5af5e/recipe/meta.yaml#L127-L137

This attached track_features entry is why MKL is chosen over OpenBLAS. MKL does not have any track_features associated with it. If there are 3 options, you would attach 0 track_features to the default, then 1 track_features for the next preferred option, and finally 2 for the least preferred option. However, since you generally only care about the one default, it is usually sufficient to add 1 track_feature to all options other than the default option.

### More info

For reference, the Visual Studio version alignment on Windows also uses mutex metapackages. https://github.com/AnacondaRecipes/aggregate/blob/9635228/vs2017/meta.yaml#L24

### Noarch packages

Noarch packages are packages that are not architecture specific and therefore only have to be built once. Noarch packages are either generic or Python. Noarch generic packages allow users to distribute docs, datasets, and source code in conda packages. Noarch Python packages are described below.

Declaring these packages as `noarch` in the `build` section of the `meta.yaml` reduces shared CI resources. Therefore, all packages that qualify to be noarch packages should be declared as such.

### Noarch Python

The `noarch: python` directive in the build section makes pure-Python packages that only need to be built once.

Noarch Python packages cut down on the overhead of building multiple different pure Python packages on different architectures and Python versions by sorting out platform and Python version-specific differences at install time.

In order to qualify as a noarch Python package, all of the following criteria must be fulfilled:

- No compiled extensions.
- No post-link, pre-link, or pre-unlink scripts.
- No OS-specific build scripts.
- No Python version-specific requirements.
- No skips except for Python version. If the recipe is py3 only, remove skip statement and add version constraint on Python in host and run section.
- 2to3 is not used.
- Scripts argument in setup.py is not used.
- If `console_script` entrypoints are in setup.py, they are listed in `meta.yaml`.
- No activate scripts.
- Not a dependency of conda.

**Note**

While `noarch: python` does not work with selectors, it does work with version constraints. `skip: True # [py2k]` can sometimes be replaced with a constrained Python version in the host and run subsections, for example: `python >=3` instead of just `python`.

**Note**

Only `console_script` entry points have to be listed in `meta.yaml`. Other entry points do not conflict with `noarch` and therefore do not require extra treatment.

Read more about conda's noarch packages.

### Link and unlink scripts

You may optionally execute scripts before and after the link and unlink steps. For more information, see Adding pre-link, post-link, and pre-unlink scripts.

### More information

For more information, go for a deeper dive in our *managing packages guide*. Learn more about package metadata, repository structure and index, and package match specifications at *Package specifications*.

### Package specification

### Package metadata

The `info/` directory contains all metadata about a package. Files in this location are not installed under the install prefix. Although you are free to add any file to this directory, conda only inspects the content of the files discussed below.

### Info

- files
  - a list of all the files in the package (not included in `info/`)
- `index.json`
  - metadata about the package including platform, version, dependencies, and build info

```
{
  "arch": "x86_64",
  "build": "py37hfa4b5c9_1",
  "build_number": 1,
  "depends": [
    "depend > 1.1.1"
  ],
  "license": "BSD 3-Clause",
  "name": "fun-packge",
  "platform": "linux",
  "subdir": "linux-64",
  "timestamp": 1535416612069,
  "version": "0.0.0"
}
```

- `paths.json`
  - a list of files in the package, along with their associated SHA-256, size in bytes, and the type of path (eg. hardlink vs. softlink)

```
{
  "paths": [
    {
      "_path": "lib/python3.7/site-packages/fun-packge/__init__.py",
```

(continues on next page)

(continued from previous page)

```
      "path_type": "hardlink",
      "sha256": "76f3b6e34feeb651aff33ca59e0279c4eadce5a50c6ad93b961c846f7ba717e9",
      "size_in_bytes": 2067
    },
    {
      "_path": "lib/python3.7/site-packages/fun-packge/__config__.py",
      "path_type": "hardlink",
      "sha256": "348e3602616c1fe4c84502b1d8cf97c740d886002c78edab176759610d287f06",
      "size_in_bytes": 87519
    },
    ...
}
```

### info/index.json

This file contains basic information about the package, such as name, version, build string, and dependencies. The content of this file is stored in `repodata.json`, which is the repository index file, hence the name `index.json`. The JSON object is a dictionary containing the keys shown below. The filename of the conda package is composed of the first 3 values, as in: `<name>-<version>-<build>.tar.bz2`.

| Key | Type | Description |
|---|---|---|
| name | string | The lowercase name of the package. May contain the "-" character. |
| version | string | The package version. May not contain "-". Conda acknowledges PEP 440. |
| build | string | The build string. May not contain "-". Differentiates builds of packages with otherwise identical names and versions, such as:<br>• A build with other dependencies, such as Python 3.4 instead of Python 2.7.<br>• A bug fix in the build process.<br>• Some different optional dependencies, such as MKL versus ATLAS linkage. Nothing in conda actually inspects the build string. Strings such as `np18py34_1` are designed only for human readability and conda never parses them. |
| build_number | integer | A non-negative integer representing the build number of the package. Unlike the build string, the build_number is inspected by conda. Conda uses it to sort packages that have otherwise identical names and versions to determine the latest one. This is important because new builds that contain bug fixes for the way a package is built may be added to a repository. |
| depends | list of strings | A list of dependency specifications, where each element is a string, as outlined in *Package match specifications*. |
| arch | string | Optional. The architecture the package is built for.<br>EXAMPLE: `x86_64`<br>Conda currently does not use this key. |
| platform | string | Optional. The OS that the package is built for.<br>EXAMPLE: `osx`<br>Conda currently does not use this key. Packages for a specific architecture and platform are usually distinguished by the repository subdirectory that contains them---see *Repository structure and index*. |

### info/files

Lists all files that are part of the package itself, 1 per line. All of these files need to get linked into the environment. Any files in the package that are not listed in this file are not linked when the package is installed. The directory delimiter for the files in `info/files` should always be "/", even on Windows. This matches the directory delimiter used in the tarball.

### info/has_prefix

Optional file. Lists all files that contain a hard-coded build prefix or placeholder prefix, which needs to be replaced by the install prefix at installation time.

> **Note**
>
> Due to the way the binary replacement works, the placeholder prefix must be longer than the install prefix.

Each line of this file should be either a path, in which case it is considered a text file with the default placeholder `/opt/anaconda1anaconda2anaconda3`, or a space-separated list of placeholder, mode, and path, where:

- Placeholder is the build or placeholder prefix.
- Mode is either `text` or `binary`.
- Path is the relative path of the file to be updated.

EXAMPLE: On Windows:

```
"Scripts/script1.py"
"C:\Users\username\anaconda\envs\_build" text "Scripts/script2.bat"
"C:/Users/username/anaconda/envs/_build" binary "Scripts/binary"
```

EXAMPLE: On macOS or Linux:

```
bin/script.sh
/Users/username/anaconda/envs/_build binary bin/binary
/Users/username/anaconda/envs/_build text share/text
```

> **Note**
>
> The directory delimiter for the relative path must always be "/", even on Windows. The placeholder may contain either "\" or "/" on Windows, but the replacement prefix will match the delimiter used in the placeholder. The default placeholder `/opt/anaconda1anaconda2anaconda3` is an exception, being replaced with the install prefix using the native path delimiter. On Windows, the placeholder and path always appear in quotes to support paths with spaces.

### info/license.txt

Optional file. The software license for the package.

### info/no_link

Optional file. Lists all files that cannot be linked - either soft-linked or hard-linked - into environments and are copied instead.

### info/about.json

Optional file. Contains the entries in the about section of the `meta.yaml` file. The following keys are added to `info/about.json` if present in the build recipe:

- home
- dev_url
- doc_url
- license_url
- license
- summary
- description
- license_family

### info/recipe

A directory containing the full contents of the build recipe.

### meta.yaml.rendered

The fully rendered build recipe. See conda render.

This directory is present only when the the include_recipe flag is `True` in the build section.

### Repository structure and index

A conda repository - or channel - is a directory tree, usually served over HTTPS, which has platform subdirectories, each of which contain conda packages and a repository index. The index file `repodata.json` lists all conda packages in the platform subdirectory. Use `conda index` to create such an index from the conda packages within a directory. It is simple mapping of the full conda package filename to the dictionary object in `info/index.json` described in link scripts.

In the following example, a repository provides the conda package `misc-1.0-np17py27_0.tar.bz2` on 64-bit Linux and 32-bit Windows:

```
<some path>/linux-64/repodata.json
                    repodata.json.bz2
                    misc-1.0-np17py27_0.tar.bz2
          /win-32/repodata.json
                  repodata.json.bz2
                  misc-1.0-np17py27_0.tar.bz2
```

**Note**

Both conda packages have identical filenames and are distinguished only by the repository subdirectory that contains them.

### Package match specifications

This match specification is not the same as the syntax used at the command line with `conda install`, such as `conda install python=3.9`. Internally, conda translates the command line syntax to the spec defined in this section.

EXAMPLE: python=3.9 is translated to python 3.9*.

Package dependencies are specified using a match specification. A match specification is a space-separated string of 1, 2, or 3 parts:

- The first part is always the exact name of the package.
- The second part refers to the version and may contain special characters:
  - | means OR.

    EXAMPLE: `1.0|1.2` matches version 1.0 or 1.2
  - * matches 0 or more characters in the version string. In terms of regular expressions, it is the same as `r.*````.

    EXAMPLE: 1.0|1.4* matches 1.0, 1.4 and 1.4.1b2, but not 1.2.
  - <, >, <=, >=, == and != are relational operators on versions, which are compared using PEP-440. For example, <=`1.0` matches `0.9`, `0.9.1`, and `1.0`, but not `1.0.1`. == and != are exact equality.

    Pre-release versioning is also supported such that >`1.0b4` will match `1.0b5` and `1.0rc1` but not `1.0b4` or `1.0a5`.

    EXAMPLE: <=1.0 matches 0.9, 0.9.1, and 1.0, but not 1.0.1.
  - , means AND.

    EXAMPLE: >=2,<3 matches all packages in the 2 series. 2.0, 2.1 and 2.9 all match, but 3.0 and 1.0 do not.
  - , has higher precedence than |, so >=1,<2|>3 means greater than or equal to 1 AND less than 2 or greater than 3, which matches 1, 1.3 and 3.0, but not 2.2.

  Conda parses the version by splitting it into parts separated by |. If the part begins with <, >, =, or !, it is parsed as a relational operator. Otherwise, it is parsed as a version, possibly containing the "*" operator.
- The third part is always the exact build string. When there are 3 parts, the second part must be the exact version.

Remember that the version specification cannot contain spaces, as spaces are used to delimit the package, version, and build string in the whole match specification. `python >= 2.7` is an invalid match specification. Furthermore, `python>=2.7` is matched as any version of a package named `python>=2.7`.

When using the command line, put double quotes around any package version specification that contains the space character or any of the following characters: <, >, *, or |.

EXAMPLE:

```
conda install numpy=1.11
conda install numpy==1.11
conda install "numpy>1.11"
conda install "numpy=1.11.1|1.11.3"
conda install "numpy>=1.8,<2"
```

### Examples

The OR constraint "numpy=1.11.1|1.11.3" matches with 1.11.1 or 1.11.3.

The AND constraint "numpy>=1.8,<2" matches with 1.8 and 1.9 but not 2.0.

The fuzzy constraint numpy=1.11 matches 1.11, 1.11.0, 1.11.1, 1.11.2, 1.11.18, and so on.

The exact constraint numpy==1.11 matches 1.11, 1.11.0, 1.11.0.0, and so on.

The build string constraint "numpy=1.11.2=*nomkl*" matches the NumPy 1.11.2 packages without MKL but not the normal MKL NumPy 1.11.2 packages.

The build string constraint "numpy=1.11.1|1.11.3=py36_0" matches NumPy 1.11.1 or 1.11.3 built for Python 3.6 but not any versions of NumPy built for Python 3.5 or Python 2.7.

The following are all valid match specifications for numpy-1.8.1-py27_0:

- numpy
- numpy 1.8*
- numpy 1.8.1
- numpy >=1.8
- numpy ==1.8.1
- numpy 1.8|1.8*
- numpy >=1.8,<2
- numpy >=1.8,<2|1.9
- numpy 1.8.1 py27_0
- numpy=1.8.1=py27_0

### Version ordering

The `class VersionOrder(object)` implements an order relation between version strings.

Version strings can contain the usual alphanumeric characters (A-Za-z0-9), separated into components by dots and underscores. Empty segments (i.e. two consecutive dots, a leading/trailing underscore) are not permitted. An optional epoch number - an integer followed by `!` - can precede the actual version string (this is useful to indicate a change in the versioning scheme itself). Version comparison is case-insensitive.

### Supported version strings

Conda supports six types of version strings:

- Release versions contain only integers, e.g. `1.0`, `2.3.5`.
- Pre-release versions use additional letters such as `a` or `rc`, for example `1.0a1`, `1.2.beta3`, `2.3.5rc3`.
- Development versions are indicated by the string `dev`, for example `1.0dev42`, `2.3.5.dev12`.
- Post-release versions are indicated by the string `post`, for example `1.0post1`, `2.3.5.post2`.
- Tagged versions have a suffix that specifies a particular property of interest, e.g. `1.1.parallel`. Tags can be added to any of the preceding 4 types. As far as sorting is concerned, tags are treated like strings in pre-release versions.
- An optional local version string separated by + can be appended to the main (upstream) version string. It is only considered in comparisons when the main versions are equal, but otherwise handled in exactly the same manner.

### Predictable version ordering

To obtain a predictable version ordering, it is crucial to keep the version number scheme of a given package consistent over time. Conda considers prerelease versions as less than release versions.

- Version strings should always have the same number of components (except for an optional tag suffix or local version string).
- Letters/Strings indicating non-release versions should always occur at the same position.

Before comparison, version strings are parsed as follows:

- They are first split into epoch, version number, and local version number at `!` and + respectively. If there is no `!`, the epoch is set to 0. If there is no +, the local version is empty.
- The version part is then split into components at `.` and `_`.
- Each component is split again into runs of numerals and non-numerals
- Subcomponents containing only numerals are converted to integers.
- Strings are converted to lowercase, with special treatment for `dev` and `post`.
- When a component starts with a letter, the fillvalue 0 is inserted to keep numbers and strings in phase, resulting in `1.1.a1' == 1.1.0a1'`.
- The same is repeated for the local version part.

Examples:

```
1.2g.beta15.rc => [[0], [1], [2, 'g'], [0, 'beta', 15], [0, 'rc']]
```

```
1!2.15.1_ALPHA => [[1], [2], [15], [1, '_alpha']]
```

The resulting lists are compared lexicographically, where the following rules are applied to each pair of corresponding subcomponents:

- Integers are compared numerically.
- Strings are compared lexicographically, case-insensitive.
- Strings are smaller than integers, except
  - `dev` versions are smaller than all corresponding versions of other types.
  - `post` versions are greater than all corresponding versions of other types.

- If a subcomponent has no correspondent, the missing correspondent is treated as integer 0 to ensure `'1.1' == 1.1.0'`.

The resulting order is:

```
  0.4
< 0.4.0
< 0.4.1.rc
== 0.4.1.RC   # case-insensitive comparison
< 0.4.1
< 0.5a1
< 0.5b3
< 0.5C1      # case-insensitive comparison
< 0.5
< 0.9.6
< 0.960923
< 1.0
< 1.1dev1    # special case ``dev``
< 1.1a1
< 1.1.0dev1  # special case ``dev``
== 1.1.dev1   # 0 is inserted before string
< 1.1.a1
< 1.1.0rc1
< 1.1.0
== 1.1
< 1.1.0post1 # special case ``post``
== 1.1.post1  # 0 is inserted before string
< 1.1post1   # special case ``post``
< 1996.07.12
< 1!0.4.1    # epoch increased
< 1!3.1.1.6
< 2!0.4.1    # epoch increased again
```

Some packages (most notably OpenSSL) have incompatible version conventions. In particular, OpenSSL interprets letters as version counters rather than pre-release identifiers. For OpenSSL, the relation `1.0.1 < 1.0.1a => True # for OpenSSL` holds, whereas conda packages use the opposite ordering. You can work around this problem by appending a dash to plain version numbers:

```
1.0.1a => 1.0.1post.a # ensure correct ordering for OpenSSL
```

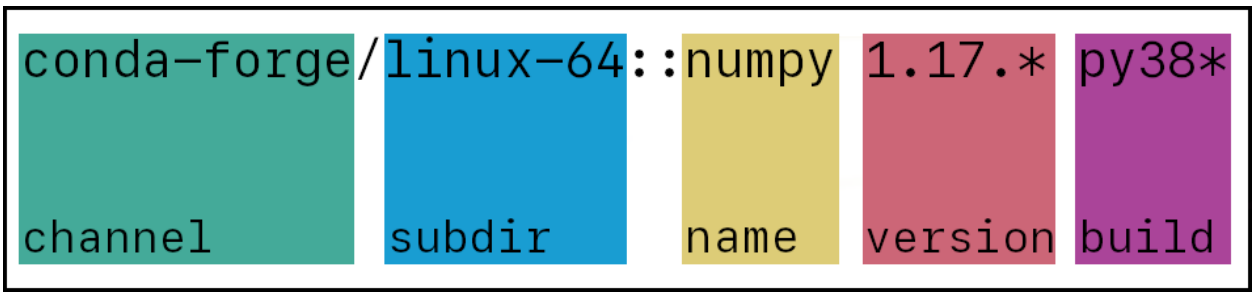
### Package search and install specifications

Conda supports the following specifications for `conda search` and `conda install`.

### Package search

`conda search` for a specific package or set of packages can be accomplished in several ways. This section includes information on the standard specification and the use of key-value pairs.

### Standard specification




**channel**
: (Optional) Can either be a channel name or URL. Channel names may include letters, numbers, dashes, and underscores.

**subdir**
: (Optional) A subdirectory of a channel. Many subdirs are used for architectures, but this is not required. Must have a channel and backslash preceding it. For example: `main/noarch`

**name**
: (Required) Package name. May include the * wildcard. For example, `*py*` returns all packages that have "py" in their names, such as "numpy", "pytorch", "python", etc.

**version**
: (Optional) Package version. May include the * wildcard or a version range(s) in single quotes. For example: `numpy=1.17.*` returns all numpy packages with a version containing "1.17." and `numpy>1.17,<1.19.2` returns all numpy packages with versions greater than 1.17 and less than 1.19.2.

**build**
: (Optional) Package build name. May include the * wildcard. For example, `numpy 1.17.3 py38*` returns all version 1.17.3 numpy packages with a build name that contains the text "py38".

### Key-value pairs

Package searches can also be performed using what is called "key-value pair notation", which has different rules than the *Standard specification* example image. The search below will return the same list of packages as the standard specification.

```
$ conda search "numpy[channel=conda-forge, subdir=linux-64, version=1.17.*, build=py38*]"
```

This notation supports the following key-value pairs:

```
- build               # validated via GlobStrMatch
- build_number        # validated via BuildNumberMatch
- channel             # validated via ChannelMatch
- features            # validated via FeatureMatch
- fn                  # validated via ExactStrMatch
```

(continues on next page)

(continued from previous page)

```
- license              # validated via CaseInsensitiveStrMatch
- license_family       # validated via CaseInsensitiveStrMatch
- md5                  # validated via ExactStrMatch
- name                 # validated via GlobLowerStrMatch
- sha256               # validated via ExactStrMatch
- subdir               # validated via ExactStrMatch
- track_features       # validated via FeatureMatch
- url                  # validated via ExactStrMatch
- version              # validated via VersionSpec
```

Key-value pair notation can be used at the same time as standard notation.

```
$ conda search "conda-forge::numpy=1.17.3[subdir=linux-64, build=py38*]"
```

**Warning**

Any search values using the key-value pair notation will override values in the rest of the search string. For example, `conda search numpy 1.17.3[version=1.19.2]` will return packages with the version `1.19.2`.

### Package installation

When you're installing packages, conda recommends being as concrete as possible. Using * wildcards and version ranges during an install will most likely cause a conflict.

However, * wildcards can still be helpful in an install command when used sparingly.

### Installing with wildcards

Let's say you are working on a project that requires version 2.3 of a package. If you upgrade to 2.4 or 3.0, your project will break. You're also using an environment file to create your environment.

In the version `2.3.1`, `2` is the major version, `3` is the minor version, and `1` is the patch. Patches typically contain bug fixes, so if you want to keep version `2.3` in your environment without updating to `2.4` or `3.0`, but want to take advantage of any bug fixes, using `2.3.*` in your environment file would be helpful to you.

### Concrete install example

Let's take the search from the *Package search* section.

```
$ conda search "conda-forge/linux-64::numpy 1.17.* py38*"
```

This returns the following:

```
Loading channels: done
# Name                       Version           Build  Channel
numpy                          1.17.3  py38h95a1406_0  conda-forge
numpy                          1.17.5  py38h18fd61f_1  conda-forge
numpy                          1.17.5  py38h95a1406_0  conda-forge
```

You can then choose a specific version and build, if necessary, and edit your `conda install` command accordingly.

```
$ conda install "conda-forge/linux-64::numpy 1.17.5 py38h95a1406_0"
```

## Channels

### What is a "channel"?

Channels are the locations where packages are stored. They serve as the base for hosting and managing packages. Conda *packages* are downloaded from remote channels, which are URLs to directories containing conda packages. The `conda` command searches a set of channels. By default, packages are automatically downloaded and updated from the default channel, which may require a paid license, as described in the repository terms of service. The `conda-forge` channel is free for all to use. You can modify which remote channels are automatically searched; this feature is beneficial when maintaining a private or internal channel. For details, see how to *modify your channel lists*.

We use conda-forge as an example channel. Conda-forge is a community channel made up of thousands of contributors. Conda-forge itself is analogous to PyPI but with a unified, automated build infrastructure and more peer review of recipes.

### Specifying channels when installing packages

- From the command line use *--channel*

```
$ conda install scipy --channel conda-forge
```

You may specify multiple channels by passing the argument multiple times:

```
$ conda install scipy --channel conda-forge --channel bioconda
```

Priority decreases from left to right - the first argument is higher priority than the second.

- From the command line use *--override-channels* to only search the specified channel(s), rather than any channels configured in .condarc. This also ignores conda's default channels.

```
$ conda search scipy --channel file:/<path to>/local-channel --override-channels
```

- In .condarc, use the key `channels` to see a list of channels for conda to search for packages.

Learn more about *managing channels*.

### Conda clone channel RSS feed

We offer a RSS feed that represents all the things that have been cloned by the channel clone and are now available behind the CDN (content delivery network). The RSS feed shows what has happened on a rolling, two-week time frame and is useful for seeing where packages are or if a sync has been run.

Let's look at the conda-forge channel RSS feed as an example.

In that feed, it will tell you every time that it runs a sync. The feed includes other entries for packages that were added or removed. Each entry is formatted to show the subdirectory the package is from, the action that was taken (addition or removal), and the name of the package. Everything has a publishing date, per standard RSS practice.

```
<rss version="0.91">
  <channel>
    <title>conda-forge updates</title>
    <link>https://anaconda.org</link>
    <description>Updates in the last two weeks</description>
    <language>en</language>
    <copyright>Copyright 2019, Anaconda, Inc.</copyright>
    <pubDate>30 Jul 2019 19:45:47 UTC</pubDate>
      <item>
        <title>running sync</title>
        <pubDate>26 Jul 2019 19:26:36 UTC</pubDate>
      </item>
      <item>
        <title>linux-64:add:jupyterlab-1.0.4-py36_0.tar.bz2</title>
        <pubDate>26 Jul 2019 19:26:36 UTC</pubDate>
      </item>
      <item>
        <title>linux-64:add:jupyterlab-1.0.4-py37_0.tar.bz2</title>
        <pubDate>26 Jul 2019 19:26:36 UTC</pubDate>
      </item>
```

### Environments

An environment is a directory that contains a specific collection of packages that you have installed. For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing. If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them. You can also share your environment with someone by giving them a copy of your `environment.yaml` file. For more information, see *Managing environments*.

### Conda directory structure

#### ROOT_DIR

The directory where the conda distribution was installed into.

EXAMPLES:

```
/opt/Anaconda  #Linux
C:\Anaconda    #Windows
```

#### /pkgs

Also referred to as PKGS_DIR. This directory contains decompressed packages, ready to be linked in conda environments. Each package resides in a subdirectory corresponding to its canonical name.

#### /envs

The system location for additional conda environments to be created.

The following subdirectories comprise the default Anaconda environment:

```
/bin
/include
/lib
/share
```

Other conda environments usually contain the same subdirectories as the default environment.

### Virtual environments

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated spaces for them that contain per-project dependencies for them.

Users can create virtual environments using one of several tools such as Pipenv or Poetry, or a conda virtual environment. Pipenv and Poetry are based around Python's built-in venv library, whereas conda has its own notion of virtual environments that is lower-level (Python itself is a dependency provided in conda environments).

Scroll to the right in the table below.

Some other traits are:

| | Python virtual environment | Conda virtual environment |
|---|---|---|
| **Libraries** | Statically link, vendor libraries in wheels, or use apt/yum/brew/etc. | Install system-level libraries as conda dependencies. |
| **System** | Depend on base system install of Python. | Python is independent from system. |
| **Extending environment** | Extend environment with pip. | Extended environment with conda or pip. |
| **Non-Python dependencies** | | Manages non-Python dependencies (R, Perl, arbitrary executables). |
| **Tracking dependencies** | | Tracks binary dependencies explicitly. |

### Why use venv-based virtual environments

- You prefer their workflow or spec formats.
- You prefer to use the system Python and libraries.
- Your project maintainers only publish to PyPI, and you prefer packages that come more directly from the project maintainers, rather than someone else providing builds based on the same code.

### Why use conda virtual environments?

- You want control over binary compatibility choices.
- You want to utilize newer language standards, such as C++ 17.
- You need libraries beyond what the system Python offers.
- You want to manage packages from languages other than Python in the same space.

### Workflow differentiators

Some questions to consider as you determine your preferred workflow and virtual environment:

- Is your environment shared across multiple code projects?
- Does your environment live alongside your code or in a separate place?
- Do your install steps involve installing any external libraries?
- Do you want to ship your environment as an archive of some sort containing the actual files of the environment?

### Package system differentiators

There are potential benefits for choosing PyPI or conda.

PyPI has one global namespace and distributed ownership of that namespace. Because of this, it is easier within PyPI to have single sources for a package directly from package maintainers.

Conda has unlimited namespaces (channels) and distributed ownership of a given channel. As such, it is easier to ensure binary compatibility within a channel using conda.

### Installing with conda

Conda packages can be installed by running the following command:

```
conda install <package>
```

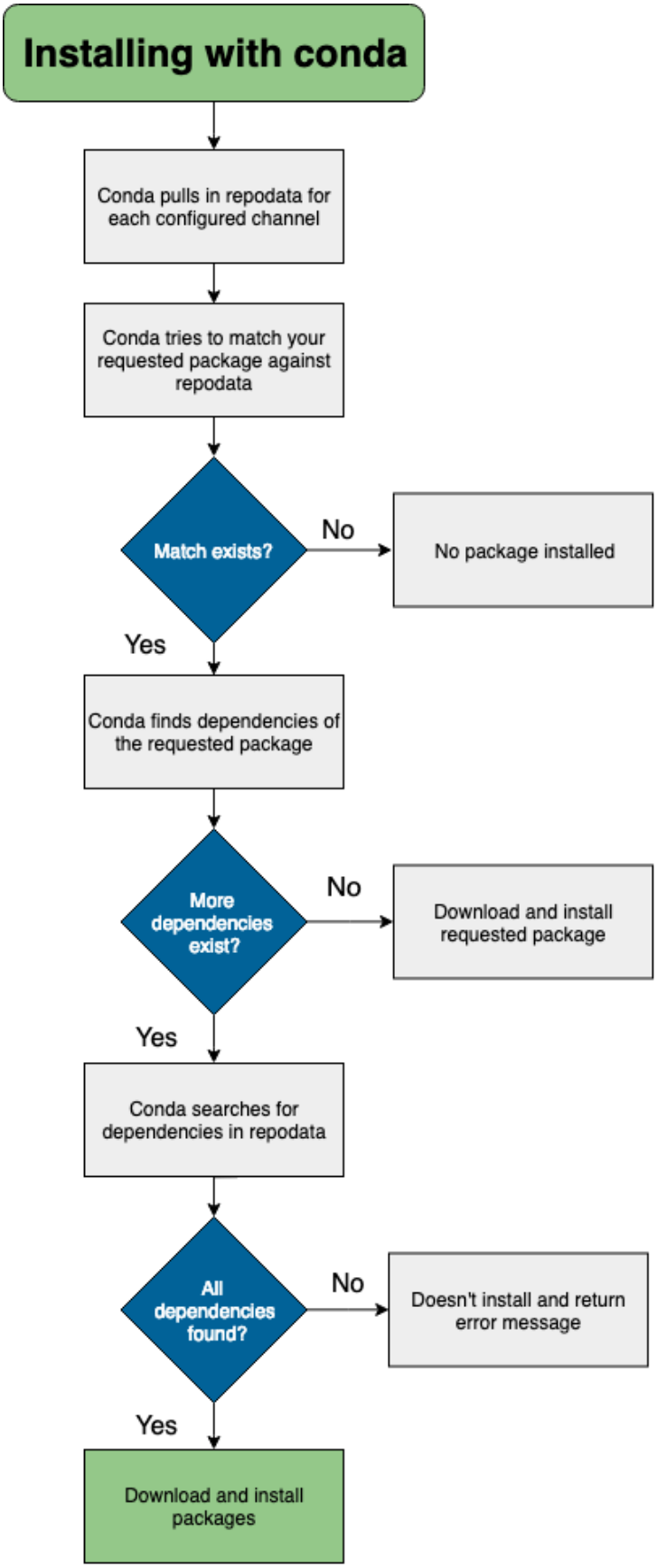
When conda installs a package, it is automatically added to your active environment. These packages are collections of files and directories that make up everything you need to use that particular library or software. For Python packages, these are primarily Python files that can be imported into other Python applications, but for compiled software packages, such as `ffmpeg`, these are typically binary executables you use directly on your computer.

> **Note**
>
> If you would like to learn more about how environments are structured, head over to *conda environments*.

Below is a more precise overview of everything that happens during the installation process for a single package:

- Currently configured channels (e.g. `defaults` or `conda-forge`) are read in order of priority
- Repodata for these configured channels is downloaded and read
- The repodata is searched for the package, starting with the highest priority channel first
- Once the package is found, conda makes a separate download request and then installs it
- This process then repeats for each of the package's dependencies, if there are any

A graphic illustration of this process is shown below:


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata
All dependencies found?
No
Doesn't install and return error message
Yes
Download and install packages

### Conda update versus conda install

`conda update` updates packages to the latest compatible version. `conda install` can be used to install any version.

Example:

- If Python 2.7.0 is currently installed, and the latest version of Python 2 is 2.7.5, then `conda update python` installs Python 2.7.5. It does not install Python 3.
- If Python 3.7.0 is currently installed, and the latest version of Python is 3.9.0, then `conda install python=3` installs Python 3.9.0.

Conda uses the same rules for other packages. `conda update` always installs the highest version with the same major version number, whereas `conda install` always installs the highest version.

### Installing conda packages offline

To install conda packages offline, run: `conda install /path-to-package/package-filename.tar.bz2/`

If you prefer, you can create a /tar/ archive file containing many conda packages and install them all with one command: `conda install /packages-path/packages-filename.tar`

> **Note**
>
> If an installed package does not work, it may be missing dependencies that need to be resolved manually.

Installing packages directly from the file does not resolve dependencies.

### Installing conda packages with a specific build number

If you want to install conda packages with the correct package specification, try `pkg_name=version=build_string`. Read more about build strings and package naming conventions. Learn more about package specifications and metadata.

For example, if you want to install llvmlite 0.31.0dev0 on Python 3.7.8, you would enter:

```
conda install  -c numba/label/dev llvmlite=0.31.0dev0=py37_8
```

### Performance

Conda's performance can be affected by a variety of things. Unlike many package managers, Anaconda's repositories generally don't filter or remove old packages from the index. This allows old environments to be easily recreated. However, it does mean that the index metadata is always growing, and thus conda becomes slower as the number of packages increases.

### How a package is installed

While you are waiting, conda is doing a lot of work installing the packages. At any point along these steps, performance issues may arise.

Conda follows these steps when installing a package:

1. Downloading and processing index metadata.
2. Reducing the index.
3. Expressing the package data and constraints as a SAT problem.
4. Running the solver.
5. Downloading and extracting packages.
6. Verifying package contents.
7. Linking packages from package cache into environments.

Therefore, if you're experiencing a slowdown, evaluate the following questions to identify potential causes:

- Are you creating a new environment or installing into an existing one?
- Does your environment have pip-installed dependencies in it?
- What channels are you using?
- What packages are you installing?
- Is the channel metadata sane?
- Are channels interacting in bad ways?

### Improving conda performance

To address these challenges, you can move packages to archive channels and follow the methods below to present conda with a smaller, simpler view than all available packages.

To speed up conda, we offer the following recommendations.

**Are you:**

- **Using conda-forge?**
  - Use conda-metachannel to reduce conda's problem size.
- **Using bioconda?**
  - Use conda-metachannel to reduce conda's problem size.
  - Read more about docker images.
- **Specifying very broad package specs?**
  - Be more specific. Letting conda filter more candidates makes it faster. For example, instead of `numpy`, we recommend `numpy=1.15` or, even better, `numpy=1.15.4`.
  - If you are using R, instead of specifying only `r-essentials`, specify `r-base=3.5 r-essentials`.
- **Feeling frustrated with "verifying transaction" and also feeling lucky?**
  - Run `conda config --set safety_checks disabled`.
- **Getting strange mixtures of defaults and conda-forge?**

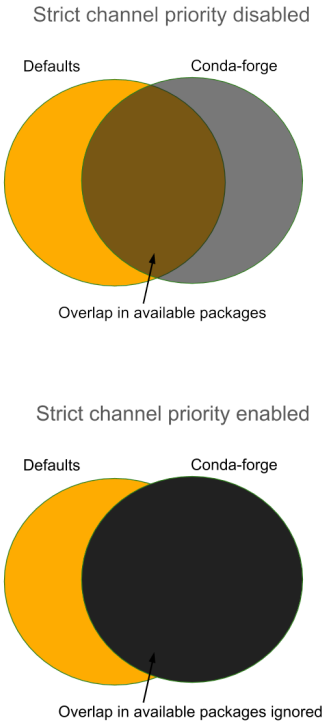
  - Run `conda config --set channel_priority strict`.
  - This also makes things go faster by eliminating possible mixed solutions.

- **Observing that an Anaconda or Miniconda installation is getting slower over time?**
  - Create a fresh environment. As environments grow, they become harder and harder to solve. Working with small, dedicated environments can be much faster.

Read more about how we made conda faster.

### Set strict channel priority

Setting strict channel priority makes it so that if a package exists on a channel, conda will ignore all packages with the same name on lower priority channels.




This can dramatically reduce package search space and reduces the use of improperly constrained packages.

One thing to consider is that setting strict channel priority may make environments unsatisfiable. Learn more about *Strict channel priority*.

### Reduce the index

One option for speeding up conda is to reduce the index. The index is reduced by conda based upon the user's input specs. It's likely that your repodata contains package data that is not used in the solving stage. Filtering out these unnecessary packages before solving can save time.

Making your input specifications more specific improves the effectiveness of the index reduction and, thus, speeds up the process. Listing a version and build string for each of your specs can dramatically reduce the number of packages that are considered when solving so that the SAT doesn't have as much work to do.

**Reducing the index:**

- Reduces unnecessary input into generating solver clauses.
- Reduces solve complexity.
- Prefers newer packages that apply constraints.

Read more on Understanding and Improving Conda's Performance.

### Conda for data scientists

Conda is useful for any packaging process but it stands out from other package and environment management systems through its utility for data science.

Conda's benefits include:

- Providing prebuilt packages which avoid the need to deal with compilers or figuring out how to set up a specific tool.
- Managing one-step installation of tools that are more challenging to install (such as TensorFlow or IRAF).
- Allowing you to provide your environment to other people across different platforms, which supports the reproducibility of research workflows.
- Allowing the use of other package management tools, such as pip, inside conda environments where a library or tools are not already packaged for conda.
- Providing commonly used data science libraries and tools, such as R, NumPy, SciPy, and TensorFlow. These are built using optimized, hardware-specific libraries (such as Intel's MKL or NVIDIA's CUDA) which speed up performance without code changes.

Read more about how conda supports data scientists.

### Plugins

In order to enable customization and extra features that are compatible with and discoverable by conda (but do not necessarily ship as a default part of the conda codebase), an official conda plugin mechanism has been implemented as of version `22.11.0`.

### Implementation

Plugins in conda integrate the "hook + entry point" structure by utilizing the Pluggy Python framework. This implementation can be broken down via the following two steps:

- Define the hook(s) to be registered
- Register the plugin under the conda entrypoint namespace

### Hook

Below is an example of a very basic plugin "hook":

Listing 1: my_plugin.py

```
import conda.plugins


@conda.plugins.hookimpl
def conda_subcommands(): ...
```

### Packaging using a pyproject.toml file

Below is an example that configures `setuptools` using a `pyproject.toml` file (note that the `setup.py` file is optional if a `pyproject.toml` file is defined, and thus will not be discussed here):

Listing 2: pyproject.toml

```
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"

[project]
name = "my-conda-plugin"
version = "1.0.0"
description = "My conda plugin"
requires-python = ">=3.7"
dependencies = ["conda"]

[project.entry-points."conda"]
my-conda-plugin = "my_plugin"
```

### Conda plugins use cases

The new conda plugin API ecosystem brings about many possibilities, including but not limited to:

- *Custom subcommands*
- Support for packaging-related topics (*e.g.*, *virtual packages*)
- Development environment integrations (*e.g.*, shells)
- Alternative dependency solver backends

- Network adapters
- Build system integrations
- Non-Python language support (*e.g.*, C, Rust)
- Experimental features that are not currently covered by conda

### Benefits of conda plugins

A conda plugin ecosystem enables contributors across the conda community to develop and share new features, thus bringing about more functionality and focus on the user experience. Though the list below is by no means exhaustive, some of the benefits of conda plugins include:

- Support for a better distribution of maintenance in the conda community
- Enabling third party contributors to use official APIs instead of having to divert to workarounds and wrappers
- The ability to extend conda internals via official APIs
- Lowering the barrier for contributions from other stakeholders in the conda ecosystem
- ... and much more!

In this section, we provide you with detailed information about the fundamental concepts in conda, including information about packages, channels, environments, and plugins, among others.

***Commands***
: Conda commands are your interface for interacting with everything

***Packages***
: Learn about the different forms a package can take

***Package specification***
: Learn about exactly what belongs in a package and what the different metadata files mean

***Package search and install***
: The search specifications for a package (for the `conda install` and `conda search` commands)

***Channels***
: Learn about channels and how they host packages

***Environments***
: Learn about how environments work and how they differ from Python's virtual environments

***Installing with conda***
: Take a deep dive into exactly what happens during an installation with conda

***Performance***
: Understand what impacts the performance and speed of conda and how to improve it

***Conda for data scientists***
: See why conda is a such a valuable tool for data scientists

***Plugins***
: The behavior of conda can be extended via plugins; learn more here

### 4.1.6 Troubleshooting

#### Using conda in Windows Batch script exits early

In conda 4.6+, the way that you interact with conda goes through a batch script (`%PREFIX%\condabin\conda.bat`). Unfortunately, this means it's a little complicated to use conda from other batch scripts. When using batch scripts from within batch scripts, you must prefix your command with `CALL`. If you do not do this, your batch script that calls conda will exit immediately after the conda usage. In other words, if you write this in a .bat file:

```
conda create myenv python
conda activate myenv
echo test
```

Neither the activation, nor the echo will happen. You must write this in your batch script:

```
CALL conda create myenv python
CALL conda activate myenv
echo test
```

This is known behavior with cmd.exe, and we have not found any way to change it. https://stackoverflow.com/questions/4798879/how-do-i-run-a-batch-script-from-within-a-batch-script/4798965

#### NumPy MKL library load failed

Error messages like

```
Intel MKL FATAL ERROR: Cannot load mkl_intel_thread.dll
```

or

```
The ordinal 241 could not be located in the the dynamic link library
```

#### Cause

NumPy is unable to load the correct MKL or Intel OpenMP runtime libraries. This is almost always caused by one of two things:

1. The environment with NumPy has not been activated.
2. Another software vendor has installed MKL or Intel OpenMP (libiomp5md.dll) files into the C:\Windows\System32 folder. These files are being loaded before Anaconda's and they're not compatible.

#### Solution

If you are not activating your environments, start with doing that. There's more info at *Activating environments*. If you are still stuck, you may need to consider more drastic measures.

1. Remove any MKL-related files from C:\Windows\System32. We recommend renaming them to add .bak to the filename to effectively hide them. Observe if any other software breaks. Try moving the DLL files alongside the .exe of the software that broke. If it works again, you can keep things in the moved state - Anaconda doesn't need MKL in System32, and no other software should need it either. If you identify software that is installing software here, please contact the creators of that software.

Inform them that their practice of installing MKL to a global location is fragile and is breaking other people's software and wasting a lot of time. See the list of guilty parties below.

2. You may try a special DLL loading mode that Anaconda builds into Python. This changes the DLL search path from System32 first to System32 as another entry on PATH, allowing libraries in your conda environment to be found before the libraries in System32. Control of this feature is done with environment variables. Only Python builds beyond these builds will react to these environment variables:

   - Python 2.7.15 build 14
   - Python 3.6.8 build 7
   - Python 3.7.2 build 8

   To update Python from the defaults channel:

   ```
   conda update -c defaults python
   ```

   **Note**

   Anaconda has built special patches into its builds of Python to enable this functionality. If you get your Python package from somewhere else (e.g. conda-forge), these flags may not do anything.

   Control environment variables:

   - `CONDA_DLL_SEARCH_MODIFICATION_ENABLE`
   - `CONDA_DLL_SEARCH_MODIFICATION_DEBUG`
   - `CONDA_DLL_SEARCH_MODIFICATION_NEVER_ADD_WINDOWS_DIRECTORY`
   - `CONDA_DLL_SEARCH_MODIFICATION_NEVER_ADD_CWD`

   To set variables on Windows, you may use either the CLI or a Windows GUI.

   - CLI: https://superuser.com/questions/79612/setting-and-getting-windows-environment-variables-from-the-command-p 79614
   - GUI: http://www.dowdandassociates.com/blog/content/howto-set-an-environment-variable-in-windows-gui/

   These should be set to a value of `1` to enable them. For example, in a terminal:

   ```
   set CONDA_DLL_SEARCH_MODIFICATION_ENABLE=1
   ```

   **Note**

   Only `CONDA_DLL_SEARCH_MODIFICATION_ENABLE` should be set finally.

List of known software that installs Intel libraries to C:\Windows\System32:

- Amplitube, by IK Multimedia
- ASIO4ALL, by Michael Tippach

If you find others, please let us know. If you're on this list and you want to fix things, let us know. In either case, the conda issue tracker at https://github.com/conda/conda/issues is the best way to reach us.

### SSL connection errors

This is a broad umbrella of errors with many causes. Here are some we've seen.

### CondaHTTPError: HTTP 000 CONNECTION FAILED

If you're on Windows and you see this error, look a little further down in the error text. Do you see something like this?:

```
SSLError(MaxRetryError('HTTPSConnectionPool(host=\'repo.anaconda.com\', port=443): Max␣
↪retries exceeded with url: /pkgs/r/win-32/repodata.json.bz2 (Caused by SSLError("Can\
↪'t connect to HTTPS URL because the SSL module is not available."))'))
```

The key part there is the last bit:

```
Caused by SSLError("Can\'t connect to HTTPS URL because the SSL module is not available.
↪")
```

Conda is having problems because it can't find the OpenSSL libraries that it needs.

### Cause

You may observe this error cropping up after a conda update. More recent versions of conda and more recent builds of Python are more strict about requiring activation of environments. We're working on better error messages for them, but here's the story for now. Windows relies on the PATH environment variable as the way to locate libraries that are not in the immediate folder, and also not in the C:\Windows\System32 folder. Searching for libraries in the PATH folders goes from left to right. If you choose to put Anaconda's folders on PATH, there are several of them:

- (install root)
- (install root)/Library/(MSYS2 env)/bin ## dependent on MSYS2 packages
- (install root)/Library/mingw-w64/bin
- (install root)/Library/usr/bin
- (install root)/Library/bin
- (install root)/Scripts
- (install root)/bin
- (install root)/condabin

Early installers for Anaconda put these on PATH. That was ultimately fragile because Anaconda isn't the only software on the system. If other software had similarly named executables or libraries, and came earlier on PATH, Anaconda could break. On the flip side, Anaconda could break other software if Anaconda were earlier in the PATH order and shadowed any other executables or libraries. To make this easier, we began recommending "activation" instead of modifying PATH. Activation is a tool where conda sets your PATH, and also runs any custom package scripts which are often used to set additional environment variables that are necessary for software to run (e.g. `JAVA_HOME`). Because activation runs only in a local terminal session (as opposed to the permanent PATH entry), it is safe to put Anaconda's PATH entries first. That means that Anaconda's libraries get higher priority when you're running Anaconda but Anaconda doesn't interfere with other software when you're not running Anaconda.

Anaconda's Python interpreter included a patch for a long time that added the (install root)/Library/bin folder to that Python's PATH. Unfortunately, this interfered with reasoning about PATH at all when using that Python interpreter. We removed that patch in Python 3.7.0, and we regret that this has caused problems for people who are not activating

their environments and who otherwise do not have the proper entries on PATH. We're experimenting with approaches that will allow our executables to be less dependent on PATH and more self-aware of their needed library load paths. For now, though, the only solutions to this problem are to manage PATH properly.

Our humble opinion is that activation is the easiest way to ensure that things work. See more information on activation in *Activating environments*.

### Solution

Use shells opened from Anaconda Navigator. If you use a GUI IDE and you see this error, ask the developers of your IDE to add activation for conda environments.

### SSL certificate errors

### Cause

Installing packages may produce a "connection failed" error if you do not have the certificates for a secure connection to the package repository.

### Solution

Pip can use the `--use-feature=truststore` option to use the operating system certificate store. This may be of help in typically corporate environments with https traffic inspection, where the corporate CA is installed in the operating system certificate store:

```
pip install --use-feature=truststore
```

Conda has a similar option:

```
conda config --set ssl_verify truststore
```

Alternatively, pip can use the `--trusted-host` option to indicate that the URL of the repository is trusted:

```
pip install --trusted-host pypi.org
```

Conda has three similar options.

1. The option `--insecure` or `-k` ignores certificate validation errors for all hosts.

   Running `conda create --help` shows:

   ```
   Networking Options:
     -k, --insecure        Allow conda to perform "insecure" SSL connections and
                           transfers. Equivalent to setting 'ssl_verify' to
                           'False'.
   ```

2. The configuration option `ssl_verify` can be set to `False`.

   Running `conda config --describe ssl_verify` shows:

   ```
   # # ssl_verify (bool, str)
   # #   aliases: verify_ssl
   # #   conda verifies SSL certificates for HTTPS requests, just like a web
   ```

(continues on next page)

(continued from previous page)

```
# #   browser. By default, SSL verification is enabled and conda operations
# #   will fail if a required URL's certificate cannot be verified. Setting
# #   ssl_verify to False disables certification verification. The value for
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true
```

Running `conda config --set ssl_verify false` modifies `~/.condarc` and sets the `-k` flag for all future conda operations performed by that user. Running `conda config --help` shows other configuration scope options.

When using `conda config`, the user's conda configuration file at `~/.condarc` is used by default. The flag `--system` will instead write to the system configuration file for all users at `<CONDA_BASE_ENV>/.condarc`. The flag `--env` will instead write to the active conda environment's configuration file at `<PATH_TO_ACTIVE_CONDA_ENV>/.condarc`. If `--env` is used and no environment is active, the user configuration file is used.

3. The configuration option `ssl_verify` can be used to install new certificates.

   Running `conda config --describe ssl_verify` shows:

```
# # ssl_verify (bool, str)
# #   aliases: verify_ssl
# #   conda verifies SSL certificates for HTTPS requests, just like a web
# #   browser. By default, SSL verification is enabled, and conda operations
# #   will fail if a required URL's certificate cannot be verified. Setting
# #   ssl_verify to False disables certification verification. The value for
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true
```

   Your network administrator can give you a certificate bundle for your network's firewall. Then `ssl_verify` can be set to the path of that certificate authority (CA) bundle and package installation operations will complete without connection errors.

   When using `conda config`, the user's conda configuration file at `~/.condarc` is used by default. The flag `--system` will instead write to the system configuration file for all users at `<CONDA_BASE_ENV>/.condarc`. The flag `--env` will instead write to the active conda environment's configuration file at `<PATH_TO_ACTIVE_CONDA_ENV>/.condarc`. If `--env` is used and no environment is active, the user configuration file is used.

### SSL verification errors

#### Cause

This error may be caused by lack of activation on Windows or expired certifications:

```
SSL verification error: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.
↪c:590)
```

#### Solution

Make sure your conda is up-to-date: `conda --version`

If not, run: `conda update conda`

Try using the operating system certificate store. Set you `ssl_verify` variable to `truststore` using the following command:

```
conda config --set ssl_verify truststore
```

If using the operating system certificate store does not solve your issue, temporarily set your `ssl_verify` variable to false, upgrade the requests package, and then set `ssl_verify` back to `true` using the following commands:

```
conda config --set ssl_verify false
conda update requests
conda config --set ssl_verify true
```

You can also set `ssl_verify` to a string path to a certificate, which can be used to verify SSL connections. Modify your `.condarc` and include the following:

```
ssl_verify: path-to-cert/chain/filename.ext
```

If the repository uses a self-signed certificate, use the actual path to the certificate. If the repository is signed by a private certificate authority (CA), the file needs to include the root certificate and any intermediate certificates.

### Permission denied errors during installation

#### Cause

The `umask` command determines the mask settings that control how file permissions are set for newly created files. If you have a very restrictive `umask`, such as `077`, you get "permission denied" errors.

#### Solution

Set a less restrictive `umask` before calling conda commands. Conda was intended as a user space tool, but often users need to use it in a global environment. One place this can go awry is with restrictive file permissions. Conda creates links when you install files that have to be read by others on the system.

To give yourself full permissions for files and directories but prevent the group and other users from having access:

1. Before installing, set the `umask` to `007`.
2. Install conda.

3. Return the `umask` to the original setting:

```
umask 007
conda install
umask 077
```

For more information on `umask`, see http://en.wikipedia.org/wiki/Umask.

### Permission denied errors after using sudo conda command

### Solution

Once you run conda with sudo, you must use sudo forever. We recommend that you NEVER run conda with sudo.

### Already installed error message

### Cause

If you are trying to fix conda problems without removing the current installation and you try to reinstall Miniconda or Anaconda to fix it, you get an error message that Miniconda or Anaconda is already installed and you cannot continue.

### Solution

Install using the `--force` option.

Download and install the appropriate Miniconda for your operating system from the Miniconda download page using the force option `--force` or `-f`:

```
bash Miniconda3-latest-MacOSX-x86_64.sh -f
```

**Note**

Substitute the appropriate filename and version for your operating system.

**Note**

Be sure that you install to the same location as your existing install so it overwrites the core conda files and does not install a duplicate in a new folder.

### Conda reports that a package is installed, but it appears not to be

Sometimes conda claims that a package is already installed but it does not appear to be, for example, a Python package that gives ImportError.

There are several possible causes for this problem, each with its own solution.

#### Cause

You are not in the same conda environment as your package.

#### Solution

1. Make sure that you are in the same conda environment as your package. The `conda info` command tells you what environment is currently active under `default environment`.
2. Verify that you are using the Python from the correct environment by running:

```
import sys

print(sys.prefix)
```

#### Cause

For Python packages, you have set the `PYTHONPATH` or `PYTHONHOME` variable. These environment variables cause Python to load files from locations other than the standard ones. Conda works best when these environment variables are not set, as their typical use cases are obviated by conda environments and a common issue is that they cause Python to pick up the wrong or broken versions of a library.

#### Solution

For Python packages, make sure you have not set the `PYTHONPATH` or `PYTHONHOME` variables. The command `conda info -a` displays the values of these environment variables.

- To unset these environment variables temporarily for the current terminal session, run `unset PYTHONPATH`.
- To unset them permanently, check for lines in the files:
  - If you use bash---`~/.bashrc`, `~/.bash_profile`, `~/.profile`.
  - If you use zsh---`~/.zshrc`.
  - If you use PowerShell on Windows, the file output by `$PROFILE`.

### Cause

You have site-specific directories or, for Python, you have so-called site-specific files. These are typically located in `~/.local` on macOS and Linux. For a full description of the locations of site-specific packages, see PEP 370. As with `PYTHONPATH`, Python may try importing packages from this directory, which can cause issues.

### Solution

For Python packages, remove site-specific directories and site-specific files.

### Cause

For C libraries, the following environment variables have been set:

- macOS---`DYLD_LIBRARY_PATH`.
- Linux---`LD_LIBRARY_PATH`.

These act similarly to `PYTHONPATH` for Python. If they are set, they can cause libraries to be loaded from locations other than the conda environment. Conda environments obviate most use cases for these variables. The command `conda info -a` shows what these are set to.

### Solution

Unset `DYLD_LIBRARY_PATH` or `LD_LIBRARY_PATH`.

### Cause

Occasionally, an installed package becomes corrupted. Conda works by unpacking the packages in the `pkgs` directory and then hard-linking them to the environment. Sometimes these get corrupted, breaking all environments that use them. They also break any additional environments since the same files are hard-linked each time.

### Solution

Run the command `conda install -f` to unarchive the package again and relink it. It also does an MD5 verification on the package. Usually if this is different it is because your channels have changed and there is a different package with the same name, version, and build number.

**Note**

This breaks the links to any other environments that already had this package installed, so you have to reinstall it there, too. It also means that running `conda install -f` a lot can use up significant disk space if you have many environments.

**Note**

The `-f` flag to `conda install` (`--force`) implies `--no-deps`, so `conda install -f package` does not reinstall any of the dependencies of `package`.

### pkg_resources.DistributionNotFound: conda==3.6.1-6-gb31b0d4-dirty

#### Cause

The local version of conda needs updating.

#### Solution

Force reinstall conda. A useful way to work off the development version of conda is to run `python setup.py develop` on a checkout of the conda GitHub repository. However, if you are not regularly running `git pull`, it is a good idea to un-develop, as you will otherwise not get any regular updates to conda. The normal way to do this is to run `python setup.py develop -u`.

However, this command does not replace the `conda` script itself. With other packages, this is not an issue, as you can just reinstall them with `conda`, but conda cannot be used if conda is installed.

The fix is to use the `./bin/conda` executable in the conda git repository to force reinstall conda. That is, run `./bin/conda install -f conda`. You can then verify with `conda info` that you have the latest version of conda, and not a git checkout. The version should not include any hashes.

### macOS error "ValueError unknown locale: UTF-8"

#### Cause

This is a bug in the macOS Terminal app that shows up only in certain locales. Locales are country-language combinations.

#### Solution

1. Open Terminal in `/Applications/Utilities`
2. Clear the Set locale environment variables on startup checkbox.

This sets your `LANG` environment variable to be empty. This may cause Terminal to use incorrect settings for your locale. The `locale` command in Terminal tells you what settings are used.

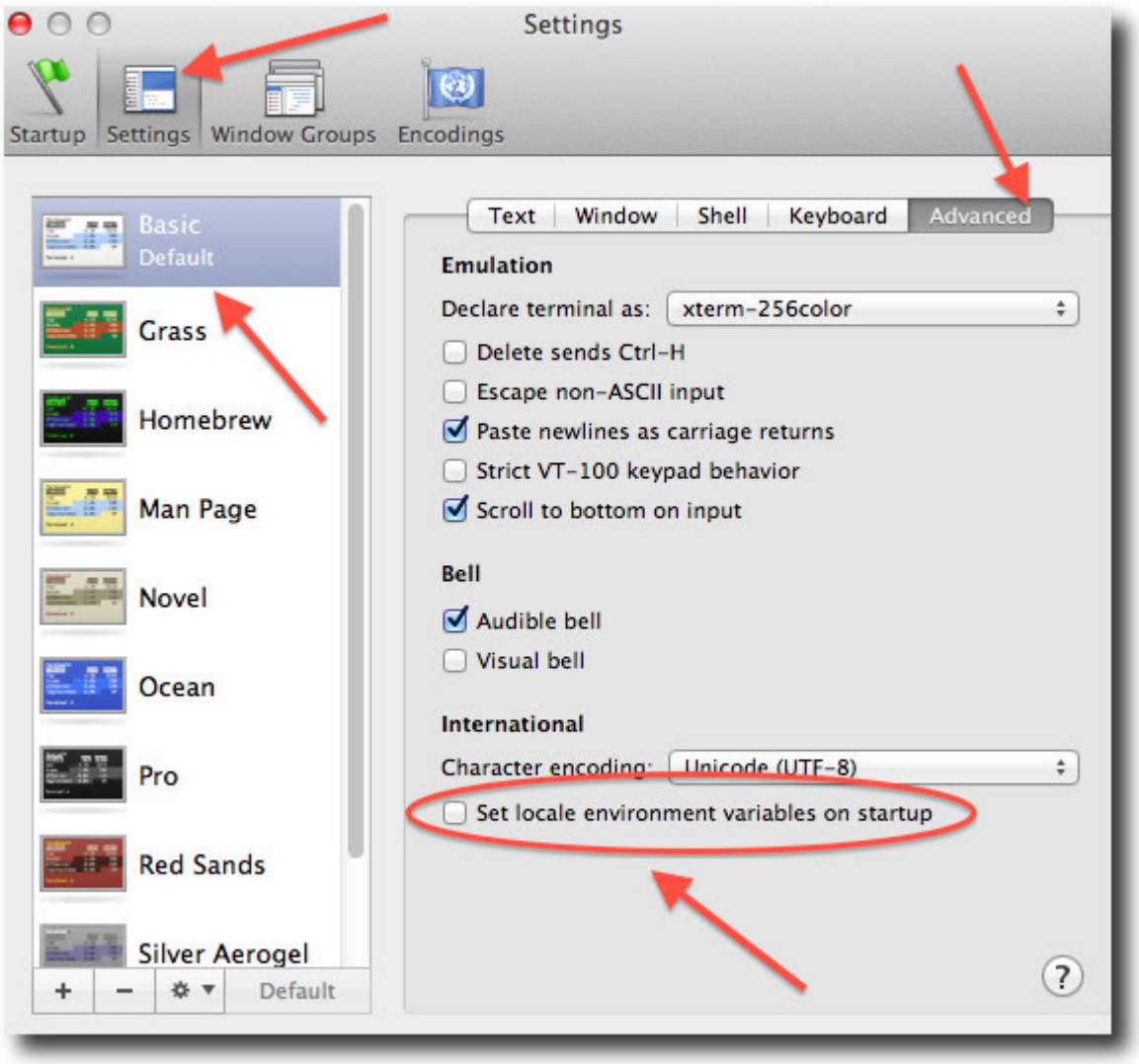
To use the correct language, add a line to your bash profile, which is typically `~/.profile`:

```
export LANG=your-lang
```

**Note**

Replace `your-lang` with the correct locale specifier for your language.

The command `locale -a` displays all the specifiers. For example, the language code for US English is `en_US.UTF-8`. The locale affects what translations are used when they are available and also how dates, currencies, and decimals are formatted.


Settings
Startup
Settings
Window Groups
Encodings
Basic
Default
Grass
Homebrew
Man Page
Novel
Ocean
Pro
Red Sands
Silver Aerogel
Default
Text
Window
Shell
Keyboard
Advanced
Emulation
Declare terminal as:
xterm-256color
Delete sends Ctrl-H
Escape non-ASCII input
Paste newlines as carriage returns
Strict VT-100 keypad behavior
Scroll to bottom on input
Bell
Audible bell
Visual bell
International
Character encoding:
Unicode (UTF-8)
Set locale environment variables on startup

### AttributeError or missing getproxies

When running a command such as `conda update ipython`, you may get an `AttributeError: 'module' object has no attribute 'getproxies'`.

### Cause

This can be caused by an old version of `requests` or by having the `PYTHONPATH` environment variable set.

### Solution

Update `requests` and be sure `PYTHONPATH` is not set:

1. Run `conda info -a` to show the `requests` version and various environment variables such as `PYTHONPATH`.
2. Update the `requests` version with `pip install -U requests`.
3. Clear `PYTHONPATH`:
    - On Windows, clear it the environment variable settings.
    - On macOS and Linux, clear it by removing it from the bash profile and restarting the shell.

### Shell commands open from the wrong location

When you run a command within a conda environment, conda does not access the correct package executable.

### Cause

In both bash and zsh, when you enter a command, the shell searches the paths in PATH one by one until it finds the command. The shell then caches the location, which is called hashing in shell terminology. When you run command again, the shell does not have to search the PATH again.

The problem is that before you installed the program, you ran a command which loaded and hashed another version of that program in some other location on the PATH, such as `/usr/bin`. Then you installed the program using `conda install`, but the shell still had the old instance hashed.

### Solution

Reactivate the environment or run `hash -r` (in bash) or `rehash` (in zsh).

When you run `conda activate`, conda automatically runs `hash -r` in bash and `rehash` in zsh to clear the hashed commands, so conda finds things in the new path on the PATH. But there is no way to do this when `conda install` is run because the command must be run inside the shell itself, meaning either you have to run the command yourself or used a source file that contains the command.

This is a relatively rare problem, since this happens only in the following circumstances:

1. You activate an environment or use the default environment, and then run a command from somewhere else.
2. Then you `conda install` a program, and then try to run the program again without running `activate` or `deactivate`.

The command `type command_name` always tells you exactly what is being run. This is better than `which command_name`, which ignores hashed commands and searches the PATH directly. The hash is reset by `conda activate` or by `hash -r` in bash or `rehash` in zsh.

### Programs fail due to invoking conda Python instead of system Python

#### Cause

After installing Anaconda or Miniconda, programs that run `python` switch from invoking the system Python to invoking the Python in the root conda environment. If these programs rely on the system Python to have certain configurations or dependencies that are not in the root conda environment Python, the programs may crash. For example, some users of the Cinnamon desktop environment on Linux Mint have reported these crashes.

#### Solution

Edit your `.bash_profile` and `.bashrc` files so that the conda binary directory, such as `~/miniconda3/bin`, is no longer added to the PATH environment variable. You can still run `conda activate` and `conda deactivate` by using their full path names, such as `~/miniconda3/bin/conda`.

You may also create a folder with symbolic links to `conda activate` and `conda deactivate` and then edit your `.bash_profile` or `.bashrc` file to add this folder to your PATH. If you do this, running `python` will invoke the system Python, but running `conda` commands, `conda activate MyEnv`, `conda activate root`, or `conda deactivate` will work normally.

After running `conda activate` to activate any environment, including after running `conda activate root`, running `python` will invoke the Python in the active conda environment.

### UnsatisfiableSpecifications error

#### Cause

Some conda package installation specifications are impossible to satisfy. For example, `conda create -n tmp python=3 wxpython=3` produces an "Unsatisfiable Specifications" error because wxPython 3 depends on Python 2.7, so the specification to install Python 3 conflicts with the specification to install wxPython 3.

When an unsatisfiable request is made to conda, conda shows a message such as this one:

```
The following specifications were found to be in conflict:
- python 3*
- wxpython 3* -> python 2.7*
Use ``conda search <package> --info`` to see the dependencies
for each package.
```

This indicates that the specification to install wxpython 3 depends on installing Python 2.7, which conflicts with the specification to install Python 3.

### Solution

Use `conda search wxpython --info` or `conda search 'wxpython=3' --info` to show information about this package and its dependencies:

```
wxpython 3.0 py27_0
--------------------
file name   : wxpython-3.0-py27_0.tar.bz2
name        : wxpython
version     : 3.0
build number: 0
build string: py27_0
channel     : defaults
size        : 34.1 MB
date        : 2014-01-10
fn          : wxpython-3.0-py27_0.tar.bz2
license_family: Other
md5         : adc6285edfd29a28224c410a39d4bdad
priority    : 2
schannel    : defaults
url         : https://repo.continuum.io/pkgs/free/osx-64/wxpython-3.0-py27_0.tar.bz2
dependencies:
    python 2.7*
    python.app
```

By examining the dependencies of each package, you should be able to determine why the installation request produced a conflict and modify the request so it can be satisfied without conflicts. In this example, you could install wxPython with Python 2.7:

```
conda create -n tmp python=2.7 wxpython=3
```

### Package installation fails from a specific channel

### Cause

Sometimes it is necessary to install a specific version from a specific channel because that version is not available from the default channel.

### Solution

The following example describes the problem in detail and its solution.

Suppose you have a specific need to install the Python `cx_freeze` module with Python 3.4. A first step is to create a Python 3.4 environment:

```
conda create -n py34 python=3.4
```

Using this environment you should first attempt:

```
conda install -n py34 cx_freeze
```

However, when you do this you get the following error:

```
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata .........
Solving package specifications: .
Error: Package missing in current osx-64 channels:
- cx_freeze

You can search for packages on anaconda.org with

  anaconda search -t conda cx_freeze
```

The message indicates that `cx_freeze` cannot be found in the default package channels. However, there may be a community-created version available and you can search for it by running the following command:

```
$ anaconda search -t conda cx_freeze
Using Anaconda Cloud api site https://api.anaconda.org
Run 'anaconda show <USER/PACKAGE>' to get more details:
Packages:
     Name                      |  Version | Package Types   | Platforms
     ------------------------- |   ------ | --------------- | ---------------
     inso/cx_freeze            |    4.3.3 | conda           | linux-64
     pyzo/cx_freeze            |    4.3.3 | conda           | linux-64, win-32, win-64,␣
↪linux-32, osx-64
                                       : http://cx-freeze.sourceforge.net/
     silg2/cx_freeze           |    4.3.4 | conda           | linux-64
                                       : create standalone executables from Python␣
↪scripts
     takluyver/cx_freeze       |    4.3.3 | conda           | linux-64
Found 4 packages
```

In this example, there are 4 different places that you could try to get the package. None of them are officially supported or endorsed by Anaconda, but members of the conda community have provided many valuable packages. If you want to go with public opinion, then the web interface provides more information:

Notice that the `pyzo` organization has by far the most downloads, so you might choose to use their package. If so, you can add their organization's channel by specifying it on the command line:

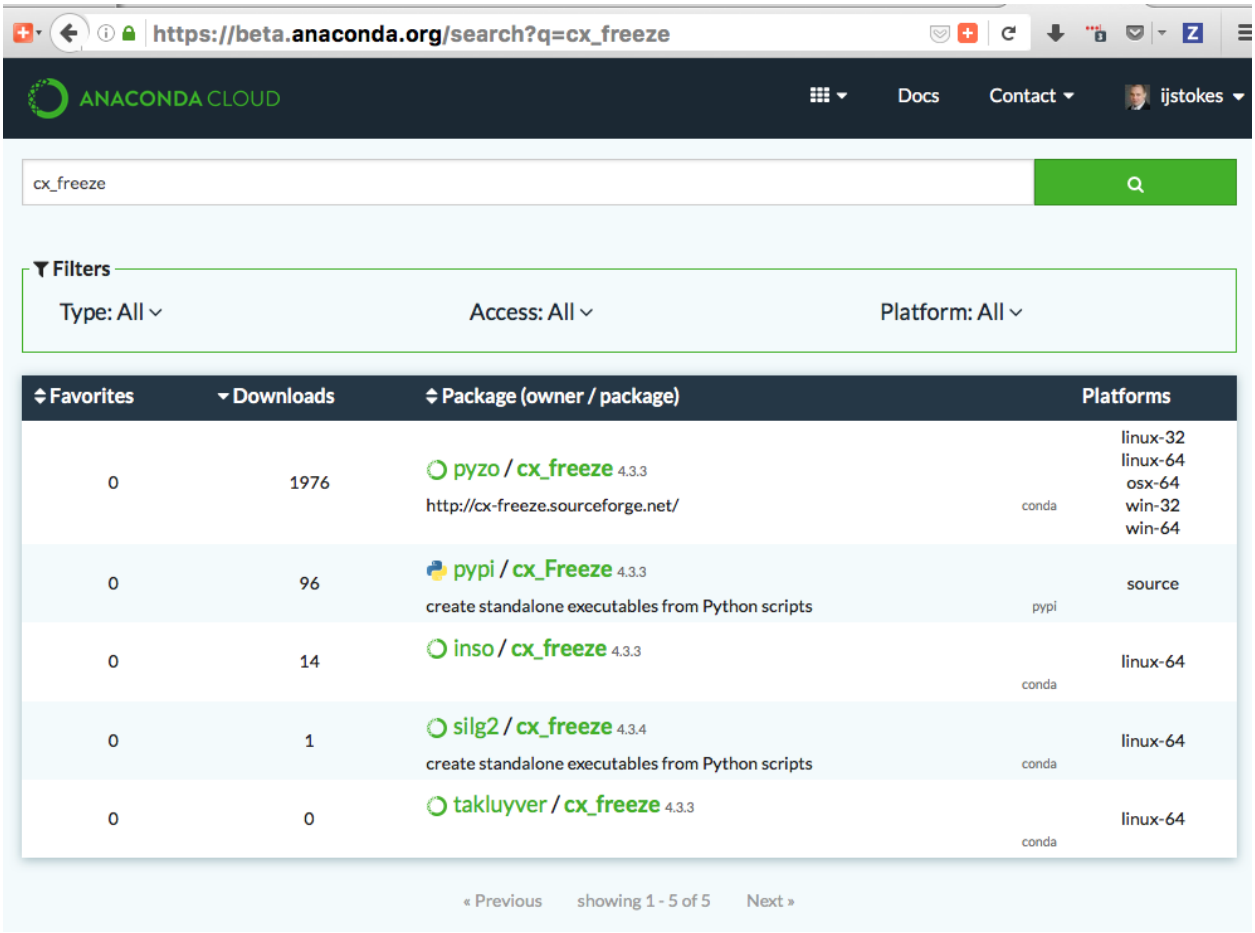
```
$ conda create -c pyzo -n cxfreeze_py34 cx_freeze python=3.4
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata: ..........
Solving package specifications: .........

Package plan for installation in environment /Users/username/anaconda/envs/cxfreeze_py34:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cx_freeze-4.3.3            |           py34_4         1.8 MB
    setuptools-20.7.0          |           py34_0         459 KB
```

(continues on next page)


https://beta.anaconda.org/search?q=cx_freeze
ANACONDA CLOUD
Docs
Contact
ijstokes
cx_freeze
Filters
Type: All
Access: All
Platform: All
Favorites
Downloads
Package (owner / package)
Platforms
0
1976
pyzo / cx_freeze 4.3.3
http://cx-freeze.sourceforge.net/
conda
linux-32
linux-64
osx-64
win-32
win-64
0
96
pypi / cx_Freeze 4.3.3
create standalone executables from Python scripts
pypi
source
0
14
inso / cx_freeze 4.3.3
conda
linux-64
0
1
silg2 / cx_freeze 4.3.4
create standalone executables from Python scripts
conda
linux-64
0
0
takluyver / cx_freeze 4.3.3
conda
linux-64
« Previous
showing 1 - 5 of 5
Next »

(continued from previous page)

```
    ------------------------------------------------------------
                                           Total:         2.3 MB

The following NEW packages will be INSTALLED:

    cx_freeze:  4.3.3-py34_4
    openssl:    1.0.2h-0
    pip:        8.1.1-py34_1
    python:     3.4.4-0
    readline:   6.2-2
    setuptools: 20.7.0-py34_0
    sqlite:     3.9.2-0
    tk:         8.5.18-0
    wheel:      0.29.0-py34_0
    xz:         5.0.5-1
    zlib:       1.2.8-0
```

Now you have a software environment sandbox created with Python 3.4 and `cx_freeze`.

### Conda automatically upgrades to unwanted version

When making a Python package for an app, you create an environment for the app from a file `req.txt` that sets a certain version, such as `python=2.7.9`. However, when you `conda install` your package, it automatically upgrades to a later version, such as `2.7.10`.

### Cause

If you make a conda package for the app using conda-build, you can set dependencies with specific version numbers. The requirements lines that say `- python` could be `- python ==2.7.9` instead. It is important to have 1 space before the == operator and no space after.

### Solution

Exercise caution when coding version requirements.

### Conda upgrade error

### Cause

Downgrading conda from 4.6.1 to 4.5.x and then trying to `conda install conda` or `conda upgrade conda` will produce a solving and upgrade error similar to the following:

```
Solving environment: failed
CondaUpgradeError: This environment has previously been operated on by a conda version␣
↪that's newer than the conda currently being used. A newer version of conda is required.
target environment location: /opt/conda
current conda version: 4.5.9
minimum conda version: 4.6
```

### Solution

Change the `.condarc` file. Set the parameter by editing the `.condarc` file directly: `allow_conda_downgrades: true` in conda version 4.5.12. This will then let you upgrade. If you have something older than 4.5.12, install conda 4.6.1 again from the package cache.

EXAMPLE: If my conda info says package cache : /opt/conda/pkgs and my Python version is 3.7, then on the command line, type `conda install /opt/conda/pkgs/conda-4.6.1-py37_0.tar.bz2` to resolve the issue.

### ValidationError: Invalid value for timestamp

### Cause

This happens when certain packages are installed with conda 4.3.28, and then conda is downgraded to 4.3.27 or earlier.

### Solution

See https://github.com/conda/conda/issues/6096.

### Unicode error after installing Python 2

Example: UnicodeDecodeError: 'ascii' codec can't decode byte 0xd3 in position 1: ordinal not in range(128)

### Cause

Python 2 is incapable of handling unicode properly, especially on Windows. In this case, if any character in your PATH env. var contains anything that is not ASCII then you see this exception.

### Solution

Remove all non-ASCII from PATH or switch to Python 3.

### Windows environment has not been activated

### Cause

You may receive a warning message if you have not activated your environment:

```
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation
```

### Solution

If you receive this warning, you need to activate your environment. To do so on Windows, on a terminal via PowerShell or the Command Prompt, run: `call <your anaconda/miniconda install location>\Scripts\activate`.

### The system cannot find the path specified on Windows

### Cause

PATH does not contain entries for all of the necessary conda directories. PATH may have too many entries from 3rd party software adding itself to PATH at install time, despite the user not needing to run the software via PATH lookup.

### Solution

Strip PATH to have fewer entries and activate your environment.

If there's some software that needs to be found on PATH (you run it via the CLI), we recommend that you create your own batch files to set PATH dynamically within a console session, rather than permanently modifying PATH in the system settings.

For example, a new conda prompt batch file that first strips PATH, then calls the correct activation procedure could look like:

```
set
PATH="%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\;<3rd-party-entries>"
call "<miniconda/anaconda root>\Scripts\activate"
```

If you need to run 3rd party software (software other than Windows built-ins and Anaconda) from this custom conda prompt, then you should add those entries (and only those strictly necessary) to the set PATH entry above. Note that only the quotes wrapping the entire expression should be there. That is how variables are properly set in batch scripts, and these account for any spaces in any entries in PATH. No additional quotes should be within the value assigned to PATH.

To make 3rd party software take precedence over the same-named programs as supplied by conda, add it to PATH after activating conda:

```
set
"PATH=%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\"
call "<miniconda/anaconda root>\Scripts\activate"
set "PATH=<3rd-party-entries>;%PATH%"
```

To make conda software take precedence, call the activation script last. Because activation prepends the conda environment PATH entries, they have priority.

```
set
PATH="%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\
↪System32\WindowsPowerShell\v1.0\;<3rd-party-entries>"
call "<miniconda/anaconda root>\Scripts\activate"
```

### 4.1.7 Cheatsheet

The conda cheatsheet contains the most important information about using conda, such as basic commands for creating and managing environments, installing packages, and importing and exporting environments.

See the `conda cheatsheet` PDF (3 MB) for a downloadable, single-page version.

**Versions**

- `conda 25.3.1` (latest)
- `conda 24.4.0`
- `conda 4.14.x`
- `conda 4.12.x`
- `conda 4.6.x`

#### Quickstart

> **Tip**
>
> It is recommended to create a new environment for any new project or workflow.

| | |
|---|---|
| verify conda install and check version | `conda info` |
| update conda in base environment | `conda update --name base conda` |
| install latest anaconda distribution (see release notes) | `conda install anaconda` |
| create a new environment (tip: name environment descriptively) | `conda create --name ENVNAME` |
| activate environment (do this before installing packages) | `conda activate ENVNAME` |

#### Channels and Packages

> **Tip**
>
> Package dependencies and platform specifics are automatically resolved when using conda.

#### Working with Conda Environments

> **Tip**
>
> List environments at the beginning of your session. Environments with an asterisk are active.

| | |
|---|---|
| list all environments and locations | `conda info --envs` |
| list all packages + source channels | `conda list --name ENVNAME --show-channel-urls` |
| install packages in environment | `conda install --name ENVNAME PKGNAME1 PKGNAME2` |
| remove package from environment | `conda uninstall --name ENVNAME PKGNAME` |
| update all packages in environment | `conda update --all --name ENVNAME` |

### Environment Management

**Tip**

Specifying the environment name confines conda commands to that environment.

| | |
|---|---|
| create environment with Python version | `conda create --name ENVNAME python=3.10` |
| clone environment | `conda create --clone ENVNAME --name NEWENV` |
| rename environment | `conda rename --name ENVNAME NEWENVNAME` |
| delete environment by name | `conda remove --name ENVNAME --all` |
| list revisions made to environment | `conda list --name ENVNAME --revisions` |
| restore environment to a revision | `conda install --name ENVNAME --revision NUMBER` |
| uninstall package from specific channel | `conda remove --name ENVNAME --channel CHANNELNAME PKGNAME` |

### Exporting Environments

**Tip**

Name your export file after your environment to preserve your environment name.

| | |
|---|---|
| cross-platform compatible | `conda export --from-history>ENV.yml` |
| platform + package specific | `conda export ENVNAME>ENV.yml` |
| platform + package + channel specific | `conda list --explicit>ENV.txt` |

### Importing Environments

**Tip**

When importing an environment, conda resolves platform and package specifics.

| | |
|---|---|
| from a .yml file | `conda env create --name ENVNAME --file ENV.yml` |
| from a .txt file | `conda create --name ENVNAME --file ENV.txt` |

#### Additional Hints

| | |
|---|---|
| get help for any command | `conda COMMAND --help` |
| get info for any package | `conda search PKGNAME --info` |
| run commands w/o user prompt, e.g., installing | `conda COMMAND ARG --yes` |
| multiple packages | `conda install PKGNAME1 PKGNAME2 --yes` |
| remove all unused files | `conda clean --all` |
| examine conda configuration | `conda config --show` |

In this guide, you will learn the about the common tasks involved with using the conda package manager.

### 4.1.8 First steps

If you are brand new to conda, then these are guides that you will want to start with first:

***Installing conda***
: Follow these instructions to get a working installation of conda on your computer

***Getting started***
: Learn the essential commands you need in your day-to-day usage of conda

***Using conda for your project***
: A tutorial explaining how to use conda in your projects

**See also**

Check out Anaconda's free course on conda basics to learn even more.

### 4.1.9 Learn more

***Configuring conda***
: Reference and explanation for all the ways you can configure conda

***Working with packages***
: Learn how to search for and use conda packages

***Working with environments***
: Learn how to create, update, remove, and export your conda environments

### 4.1.10 Additional resources

***Cheat sheet***
: Commonly used commands organized into a PDF

***Troubleshooting***
: Various solutions to commonly encountered problems

## 4.2 Configuration

```
# ##############################################################
# ##              Channel Configuration                       ##
# ##############################################################

# # channels (sequence: primitive)
# #   aliases: channel
# #   env var string delimiter: ','
# #   The list of conda channels to include for relevant operations.
# #
# channels: []

# # channel_alias (str)
# #   The prepended url location to associate with channel names.
# #
# channel_alias: https://conda.anaconda.org

# # channel_settings (sequence: map)
# #   env var string delimiter: ','
# #   A list of mappings that allows overriding certain settings for a
# #   single channel. Each list item should include at least the "channel"
# #   key and the setting you would like to override.
# #
# channel_settings: []

# # default_channels (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of channel names and/or urls used for the 'defaults'
# #   multichannel.
# #
# default_channels:
#   - https://repo.anaconda.com/pkgs/main
#   - https://repo.anaconda.com/pkgs/r

# # override_channels_enabled (bool)
# #   Permit use of the --override-channels command-line flag.
# #
# override_channels_enabled: true

# # allowlist_channels (sequence: primitive)
# #   aliases: whitelist_channels
# #   env var string delimiter: ','
# #   The exclusive list of channels allowed to be used on the system. Use
# #   of any other channels will result in an error. If conda-build channels
# #   are to be allowed, along with the --use-local command line flag, be
# #   sure to include the 'local' channel in the list. If the list is empty
# #   or left undefined, no channel exclusions will be enforced.
# #
# allowlist_channels: []

# # denylist_channels (sequence: primitive)
```

(continues on next page)

(continued from previous page)

```
# #   env var string delimiter: ','
# #   The list of channels that are denied to be used on the system. Use of
# #   any of these channels will result in an error. If conda-build channels
# #   are to be allowed, along with the --use-local command line flag, be
# #   sure to not include the 'local' channel in the list. If the list is
# #   empty or left undefined, no channel exclusions will be enforced.
# #
# denylist_channels: []

# # custom_channels (map: primitive)
# #   A map of key-value pairs where the key is a channel name and the value
# #   is a channel location. Channels defined here override the default
# #   'channel_alias' value. The channel name (key) is not included in the
# #   channel location (value).  For example, to override the location of
# #   the 'conda-forge' channel where the url to repodata is
# #   https://anaconda-repo.dev/packages/conda-forge/linux-64/repodata.json,
# #   add an entry 'conda-forge: https://anaconda-repo.dev/packages'.
# #
# custom_channels:
#   pkgs/pro: https://repo.anaconda.com

# # custom_multichannels (map: sequence)
# #   A multichannel is a metachannel composed of multiple channels. The two
# #   reserved multichannels are 'defaults' and 'local'. The 'defaults'
# #   multichannel is customized using the 'default_channels' parameter. The
# #   'local' multichannel is a list of file:// channel locations where
# #   conda-build stashes successfully-built packages.  Other multichannels
# #   can be defined with custom_multichannels, where the key is the
# #   multichannel name and the value is a list of channel names and/or
# #   channel urls.
# #
# custom_multichannels: {}

# # migrated_channel_aliases (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of previously-used channel_alias values. Useful when switching
# #   between different Anaconda Repository instances.
# #
# migrated_channel_aliases: []

# # migrated_custom_channels (map: primitive)
# #   A map of key-value pairs where the key is a channel name and the value
# #   is the previous location of the channel.
# #
# migrated_custom_channels: {}

# # add_anaconda_token (bool)
# #   aliases: add_binstar_token
# #   In conjunction with the anaconda command-line client (installed with
# #   `conda install anaconda-client`), and following logging into an
# #   Anaconda Server API site using `anaconda login`, automatically apply a
# #   matching private token to enable access to private packages and
```

(continues on next page)

(continued from previous page)

```
# #   channels.
# #
# add_anaconda_token: true

# # allow_non_channel_urls (bool)
# #   Warn, but do not fail, when conda detects a channel url is not a valid
# #   channel.
# #
# allow_non_channel_urls: false

# # restore_free_channel (bool)
# #   "                  Add the "free" channel back into defaults, behind
# #   "main" in priority. The "free"                 channel was removed
# #   from the collection of default channels in conda 4.7.0.
# #
# restore_free_channel: false

# # repodata_fns (sequence: primitive)
# #   env var string delimiter: ','
# #   Specify filenames for repodata fetching. The default is
# #   ('current_repodata.json', 'repodata.json'), which tries a subset of
# #   the full index containing only the latest version for each package,
# #   then falls back to repodata.json.  You may want to specify something
# #   else to use an alternate index that has been reduced somehow.
# #
# repodata_fns:
#   - current_repodata.json
#   - repodata.json

# # use_only_tar_bz2 (NoneType, bool)
# #   A boolean indicating that only .tar.bz2 conda packages should be
# #   downloaded. This is forced to True if conda-build is installed and
# #   older than 3.18.3, because older versions of conda break when conda
# #   feeds it the new file format.
# #
# use_only_tar_bz2:

# # repodata_threads (int)
# #   Threads to use when downloading and reading repodata.  When not set,
# #   defaults to None, which uses the default ThreadPoolExecutor behavior.
# #
# repodata_threads: 0

# # fetch_threads (int)
# #   Threads to use when downloading packages.  When not set, defaults to
# #   None, which uses the default ThreadPoolExecutor behavior.
# #
# fetch_threads: 0

# # experimental (sequence: primitive)
# #   env var string delimiter: ','
# #   List of experimental features to enable.
```

(continues on next page)

(continued from previous page)

```
# #
# experimental: []

# # no_lock (bool)
# #   Disable index cache lock (defaults to enabled).
# #
# no_lock: false

# # repodata_use_zst (bool)
# #   Disable check for `repodata.json.zst`; use `repodata.json` only.
# #
# repodata_use_zst: true


# ######################################################
# ##            Basic Conda Configuration             ##
# ######################################################

# # envs_dirs (sequence: primitive)
# #   aliases: envs_path
# #   env var string delimiter: ':'
# #   The list of directories to search for named environments. When
# #   creating a new named environment, the environment will be placed in
# #   the first writable location.
# #
# envs_dirs: []

# # pkgs_dirs (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of directories where locally-available packages are linked
# #   from at install time. Packages not locally available are downloaded
# #   and extracted into the first writable directory.
# #
# pkgs_dirs: []

# # default_threads (int)
# #   Threads to use by default for parallel operations.  Default is None,
# #   which allows operations to choose themselves.  For more specific
# #   control, see the other *_threads parameters:     * repodata_threads -
# #   for fetching/loading repodata     * verify_threads - for verifying
# #   package contents in transactions     * execute_threads - for carrying
# #   out the unlinking and linking steps
# #
# default_threads: 0


# ######################################################
# ##              Network Configuration               ##
# ######################################################

# # client_ssl_cert (NoneType, str)
# #   aliases: client_cert
```

(continues on next page)

(continued from previous page)

```
# #   A path to a single file containing a private key and certificate (e.g.
# #   .pem file). Alternately, use client_ssl_cert_key in conjunction with
# #   client_ssl_cert for individual files.
# #
# client_ssl_cert:

# # client_ssl_cert_key (NoneType, str)
# #   aliases: client_cert_key
# #   Used in conjunction with client_ssl_cert for a matching key file.
# #
# client_ssl_cert_key:

# # local_repodata_ttl (bool, int)
# #   For a value of False or 0, always fetch remote repodata (HTTP 304
# #   responses respected). For a value of True or 1, respect the HTTP
# #   Cache-Control max-age header. Any other positive integer values is the
# #   number of seconds to locally cache repodata before checking the remote
# #   server for an update.
# #
# local_repodata_ttl: 1

# # offline (bool)
# #   Restrict conda to cached download content and file:// based urls.
# #
# offline: false

# # proxy_servers (map: primitive)
# #   A mapping to enable proxy settings. Keys can be either (1) a
# #   scheme://hostname form, which will match any request to the given
# #   scheme and exact hostname, or (2) just a scheme, which will match
# #   requests to that scheme. Values are are the actual proxy server, and
# #   are of the form 'scheme://[user:password@]host[:port]'. The optional
# #   'user:password' inclusion enables HTTP Basic Auth with your proxy.
# #
# proxy_servers: {}

# # remote_connect_timeout_secs (float)
# #   The number seconds conda will wait for your client to establish a
# #   connection to a remote url resource.
# #
# remote_connect_timeout_secs: 9.15

# # remote_max_retries (int)
# #   The maximum number of retries each HTTP connection should attempt.
# #
# remote_max_retries: 3

# # remote_backoff_factor (int)
# #   The factor determines the time HTTP connection should wait for
# #   attempt.
# #
# remote_backoff_factor: 1
```

(continues on next page)

(continued from previous page)

```
# # remote_read_timeout_secs (float)
# #   Once conda has connected to a remote resource and sent an HTTP
# #   request, the read timeout is the number of seconds conda will wait for
# #   the server to send a response.
# #
# remote_read_timeout_secs: 60.0

# # ssl_verify (bool, str)
# #   aliases: verify_ssl
# #   Conda verifies SSL certificates for HTTPS requests, just like a web
# #   browser. By default, SSL verification is enabled, and conda operations
# #   will fail if a required url's certificate cannot be verified. Setting
# #   ssl_verify to False disables certification verification. The value for
# #   ssl_verify can also be (1) a path to a CA bundle file, (2) a path to a
# #   directory containing certificates of trusted CA, or (3) 'truststore'
# #   to use the operating system certificate store.
# #
# ssl_verify: true


# ######################################################
# ##               Solver Configuration               ##
# ######################################################

# # aggressive_update_packages (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of packages that, if installed, are always updated to the
# #   latest possible version.
# #
# aggressive_update_packages:
#   - ca-certificates
#   - certifi
#   - openssl

# # auto_update_conda (bool)
# #   aliases: self_update
# #   Automatically update conda when a newer or higher priority version is
# #   detected.
# #
# auto_update_conda: true

# # channel_priority (ChannelPriority)
# #   Accepts values of 'strict', 'flexible', and 'disabled'. The default
# #   value is 'flexible'. With strict channel priority, packages in lower
# #   priority channels are not considered if a package with the same name
# #   appears in a higher priority channel. With flexible channel priority,
# #   the solver may reach into lower priority channels to fulfill
# #   dependencies, rather than raising an unsatisfiable error. With channel
# #   priority disabled, package version takes precedence, and the
# #   configured priority of channels is used only to break ties. In
# #   previous versions of conda, this parameter was configured as either
```

(continues on next page)

(continued from previous page)

```
# #   True or False. True is now an alias to 'flexible'.
# #
# channel_priority: flexible

# # create_default_packages (sequence: primitive)
# #   env var string delimiter: ','
# #   Packages that are by default added to a newly created environments.
# #
# create_default_packages: []

# # disallowed_packages (sequence: primitive)
# #   aliases: disallow
# #   env var string delimiter: '&'
# #   Package specifications to disallow installing. The default is to allow
# #   all packages.
# #
# disallowed_packages: []

# # force_reinstall (bool)
# #   Ensure that any user-requested package for the current operation is
# #   uninstalled and reinstalled, even if that package already exists in
# #   the environment.
# #
# force_reinstall: false

# # pinned_packages (sequence: primitive)
# #   env var string delimiter: '&'
# #   A list of package specs to pin for every environment resolution. This
# #   parameter is in BETA, and its behavior may change in a future release.
# #
# pinned_packages: []

# # prefix_data_interoperability (bool)
# #   aliases: p, i, p, _, i, n, t, e, r, o, p, _, e, n, a, b, l, e, d
# #   Enable plugins to allow conda to interact with non-conda-installed
# #   packages.
# #
# prefix_data_interoperability: false

# # track_features (sequence: primitive)
# #   env var string delimiter: ','
# #   A list of features that are tracked by default. An entry here is
# #   similar to adding an entry to the create_default_packages list.
# #
# track_features: []

# # solver (str)
# #   aliases: experimental_solver
# #   A string to choose between the different solver logics implemented in
# #   conda. A solver logic takes care of turning your requested packages
# #   into a list of specs to add and/or remove from a given environment,
# #   based on their dependencies and specified constraints.
```

(continues on next page)

(continued from previous page)

```
# #
# solver: libmamba


# ######################################################
# ##  Package Linking and Install-time Configuration  ##
# ######################################################

# # allow_softlinks (bool)
# #   When allow_softlinks is True, conda uses hard-links when possible, and
# #   soft-links (symlinks) when hard-links are not possible, such as when
# #   installing on a different filesystem than the one that the package
# #   cache is on. When allow_softlinks is False, conda still uses hard-
# #   links when possible, but when it is not possible, conda copies files.
# #   Individual packages can override this setting, specifying that certain
# #   files should never be soft-linked (see the no_link option in the build
# #   recipe documentation).
# #
# allow_softlinks: false

# # always_copy (bool)
# #   aliases: copy
# #   Register a preference that files be copied into a prefix during
# #   install rather than hard-linked.
# #
# always_copy: false

# # always_softlink (bool)
# #   aliases: softlink
# #   Register a preference that files be soft-linked (symlinked) into a
# #   prefix during install rather than hard-linked. The link source is the
# #   'pkgs_dir' package cache from where the package is being linked.
# #   WARNING: Using this option can result in corruption of long-lived
# #   conda environments. Package caches are *caches*, which means there is
# #   some churn and invalidation. With this option, the contents of
# #   environments can be switched out (or erased) via operations on other
# #   environments.
# #
# always_softlink: false

# # path_conflict (PathConflict)
# #   The method by which conda handle's conflicting/overlapping paths
# #   during a create, install, or update operation. The value must be one
# #   of 'clobber', 'warn', or 'prevent'. The '--clobber' command-line flag
# #   or clobber configuration parameter overrides path_conflict set to
# #   'prevent'.
# #
# path_conflict: clobber

# # rollback_enabled (bool)
# #   Should any error occur during an unlink/link transaction, revert any
# #   disk mutations made to that point in the transaction.
```

(continues on next page)

(continued from previous page)

```
# #
# rollback_enabled: true

# # safety_checks (SafetyChecks)
# #   Enforce available safety guarantees during package installation. The
# #   value must be one of 'enabled', 'warn', or 'disabled'.
# #
# safety_checks: warn

# # extra_safety_checks (bool)
# #   Spend extra time validating package contents.  Currently, runs sha256
# #   verification on every file within each package during installation.
# #
# extra_safety_checks: false

# # signing_metadata_url_base (NoneType, str)
# #   Base URL for obtaining trust metadata updates (i.e., the `*.root.json`
# #   and `key_mgr.json` files) used to verify metadata and (eventually)
# #   package signatures.
# #
# signing_metadata_url_base:

# # shortcuts (bool)
# #   Allow packages to create OS-specific shortcuts (e.g. in the Windows
# #   Start Menu) at install time.
# #
# shortcuts: true

# # shortcuts_only (sequence: primitive)
# #   env var string delimiter: ','
# #   Create shortcuts only for the specified package names.
# #
# shortcuts_only: []

# # non_admin_enabled (bool)
# #   Allows completion of conda's create, install, update, and remove
# #   operations, for non-privileged (non-root or non-administrator) users.
# #
# non_admin_enabled: true

# # separate_format_cache (bool)
# #   Treat .tar.bz2 files as different from .conda packages when filenames
# #   are otherwise similar. This defaults to False, so that your package
# #   cache doesn't churn when rolling out the new package format. If you'd
# #   rather not assume that a .tar.bz2 and .conda from the same place
# #   represent the same content, set this to True.
# #
# separate_format_cache: false

# # verify_threads (int)
# #   Threads to use when performing the transaction verification step.
# #   When not set, defaults to 1.
```

(continues on next page)

(continued from previous page)

```
# #
# verify_threads: 0

# # execute_threads (int)
# #   Threads to use when performing the unlink/link transaction.  When not
# #   set, defaults to 1.  This step is pretty strongly I/O limited, and you
# #   may not see much benefit here.
# #
# execute_threads: 0


# ##########################################################
# ##            Conda-build Configuration             ##
# ##########################################################

# # bld_path (str)
# #   The location where conda-build will put built packages. Same as
# #   'croot', but 'croot' takes precedence when both are defined. Also used
# #   in construction of the 'local' multichannel.
# #
# bld_path: ''

# # croot (str)
# #   The location where conda-build will put built packages. Same as
# #   'bld_path', but 'croot' takes precedence when both are defined. Also
# #   used in construction of the 'local' multichannel.
# #
# croot: ''

# # anaconda_upload (NoneType, bool)
# #   aliases: binstar_upload
# #   Automatically upload packages built with conda build to anaconda.org.
# #
# anaconda_upload:

# # conda_build (map: primitive)
# #   aliases: conda-build
# #   General configuration parameters for conda-build.
# #
# conda_build: {}


# ##########################################################
# ##  Output, Prompt, and Flow Control Configuration  ##
# ##########################################################

# # always_yes (NoneType, bool)
# #   aliases: yes
# #   Automatically choose the 'yes' option whenever asked to proceed with a
# #   conda operation, such as when running `conda install`.
# #
# always_yes:
```

(continues on next page)

(continued from previous page)

```
# # auto_activate (bool)
# #   aliases: auto_activate_base
# #   Automatically activate the environment given at
# #   'default_activation_env' during shell initialization.
# #
# auto_activate: true

# # default_activation_env (str)
# #   The environment to be automatically activated on startup if
# #   'auto_activate' is True. Also sets the default environment to activate
# #   when 'conda activate' receives no arguments.
# #
# default_activation_env: base

# # auto_stack (int)
# #   Implicitly use --stack when using activate if current level of nesting
# #   (as indicated by CONDA_SHLVL environment variable) is less than or
# #   equal to specified value. 0 or false disables automatic stacking, 1 or
# #   true enables it for one level.
# #
# auto_stack: 0

# # changeps1 (bool)
# #   When using activate, change the command prompt ($PS1) to include the
# #   activated environment.
# #
# changeps1: true

# # env_prompt (str)
# #   Template for prompt modification based on the active environment.
# #   Currently supported template variables are '{prefix}', '{name}', and
# #   '{default_env}'. '{prefix}' is the absolute path to the active
# #   environment. '{name}' is the basename of the active environment
# #   prefix. '{default_env}' holds the value of '{name}' if the active
# #   environment is a conda named environment ('-n' flag), or otherwise
# #   holds the value of '{prefix}'. Templating uses python's str.format()
# #   method.
# #
# env_prompt: '({default_env}) '

# # json (bool)
# #   Ensure all output written to stdout is structured json.
# #
# json: false

# # console (str)
# #   Configure different backends to be used while rendering normal console
# #   output. Defaults to "classic".
# #
# console: classic
```

(continues on next page)

(continued from previous page)

```
# # notify_outdated_conda (bool)
# #   Notify if a newer version of conda is detected during a create,
# #   install, update, or remove operation.
# #
# notify_outdated_conda: true

# # quiet (bool)
# #   Disable progress bar display and other output.
# #
# quiet: false

# # report_errors (NoneType, bool)
# #   Opt in, or opt out, of automatic error reporting to core maintainers.
# #   Error reports are anonymous, with only the error stack trace and
# #   information given by `conda info` being sent.
# #
# report_errors:

# # show_channel_urls (NoneType, bool)
# #   Show channel URLs when displaying what is going to be downloaded.
# #
# show_channel_urls:

# # list_fields (sequence: primitive)
# #   env var string delimiter: ','
# #   Default fields to report as columns in the output of `conda list`.
# #
# list_fields:
#   - name
#   - version
#   - build
#   - channel_name

# # verbosity (int)
# #   aliases: verbose
# #   Sets output log level. 0 is warn. 1 is info. 2 is debug. 3 is trace.
# #
# verbosity: 0

# # unsatisfiable_hints (bool)
# #   A boolean to determine if conda should find conflicting packages in
# #   the case of a failed install.
# #
# unsatisfiable_hints: true

# # unsatisfiable_hints_check_depth (int)
# #   An integer that specifies how many levels deep to search for
# #   unsatisfiable dependencies. If this number is 1 it will complete the
# #   unsatisfiable hints fastest (but perhaps not the most complete). The
# #   higher this number, the longer the generation of the unsat hint will
# #   take. Defaults to 3.
# #
```

(continues on next page)

(continued from previous page)

```
# unsatisfiable_hints_check_depth: 2

# # number_channel_notices (int)
# #   Sets the number of channel notices to be displayed when running
# #   commands the "install", "create", "update", "env create", and "env
# #   update" . Defaults to 5. In order to completely suppress channel
# #   notices, set this to 0.
# #
# number_channel_notices: 5

# # envvars_force_uppercase (bool)
# #   Force uppercase for new environment variable names. Defaults to True.
# #
# envvars_force_uppercase: true


# ######################################################
# ##               Plugin Configuration               ##
# ######################################################

# # no_plugins (bool)
# #   Disable all currently-registered plugins, except built-in conda
# #   plugins.
# #
# no_plugins: false


# ######################################################
# ##                    Experimental                  ##
# ######################################################

# # environment_specifier (NoneType, str)
# #   aliases: env_spec
# #   **EXPERIMENTAL** While experimental, expect both major and minor
# #   changes across minor releases.  The name of the environment specifier
# #   plugin that should be used for this context. If not specified, the
# #   plugin manager will try to detect the plugin to use.
# #
# environment_specifier:
```

## 4.3 Commands

Conda provides many commands for managing packages and environments. The links on this page provide help for each command. You can also access help from the command line with the `--help` flag:

```
conda install --help
```

The following commands are part of conda:

### 4.3.1 conda activate

```
usage: conda activate [-h]
```

### 4.3.2 conda clean

Remove unused packages and caches.

```
usage: conda clean [-h] [-a] [-i] [-p] [-t] [-f] [-c [TEMPFILES ...]] [-l]
                   [--json] [--console CONSOLE] [-v] [-q] [-d] [-y]
```

#### Removal Targets

| | |
|---|---|
| **-a, --all** | Remove index cache, lock files, unused cache packages, tarballs, and logfiles. |
| **-i, --index-cache** | Remove index cache. |
| **-p, --packages** | Remove unused packages from writable package caches. WARNING: This does not check for packages installed using symlinks back to the package cache. |
| **-t, --tarballs** | Remove cached package tarballs. |
| **-f, --force-pkgs-dirs** | Remove *all* writable package caches. This option is not included with the --all flag. WARNING: This will break environments with packages installed using symlinks back to the package cache. |
| **-c, --tempfiles** | Remove temporary files that could not be deleted earlier due to being in-use. The argument for the --tempfiles flag is a path (or list of paths) to the environment(s) where the tempfiles should be found and removed. |
| **-l, --logfiles** | Remove log files. |

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |
| **-d, --dry-run** | Only display what would have been done. |
| **-y, --yes** | Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc. |

Examples:

```
conda clean --tarballs
```

### 4.3.3 conda compare

Compare packages between conda environments.

```
usage: conda compare [-h] [--json] [--console CONSOLE] [-v] [-q]
                     [-n ENVIRONMENT | -p PATH]
                     file
```

#### Positional Arguments

| | |
|---|---|
| **file** | Path to the environment file that is to be compared against. |

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

#### Target Environment Specification

| | |
|---|---|
| **-n, --name** | Name of environment. |
| **-p, --prefix** | Full path to environment location (i.e. prefix). |

Examples:

Compare packages in the current environment with respect to 'environment.yml' located in the current working directory:

```
conda compare environment.yml
```

Compare packages installed into the environment 'myenv' with respect to 'environment.yml' in a different directory:

```
conda compare -n myenv path/to/file/environment.yml
```

### 4.3.4 conda config

Modify configuration values in .condarc.

This is modeled after the git config command. Writes to the user .condarc file (/home/docs/.condarc) by default. Use the --show-sources flag to display all identified configuration locations on your computer.

```
usage: conda config [-h] [--json] [--console CONSOLE] [-v] [-q]
                    [--system | --env | --file FILE | -n ENVIRONMENT | -p PATH]
                    [--show [SHOW ...] | --show-sources | --validate |
                    --describe [DESCRIBE ...] | --write-default]
                    [--get [KEY ...] | --append KEY VALUE | --prepend KEY
                    VALUE | --set KEY VALUE | --remove KEY VALUE |
                    --remove-key KEY | --stdin]
```

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

### Config File Location Selection

Without one of these flags, the user config file at '/home/docs/.condarc' is used.

**--system** Write to the system .condarc file at '/home/docs/checkouts/readthedocs.org/user_builds/continuumio-conda/conda/latest/.condarc'.

**--env** Write to the active conda environment .condarc file (<no active environment>). If no environment is active, write to the user config file (/home/docs/.condarc).

**--file** Write to the given file.

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Config Subcommands

**--show** Display configuration values as calculated and compiled. If no arguments given, show information for all configuration values.

**--show-sources** Display all identified configuration sources.

**--validate** Validate all configuration sources. Iterates over all .condarc files and checks for parsing errors.

**--describe** Describe given configuration parameters. If no arguments given, show information for all configuration parameters.

**--write-default** Write the default configuration to a file. Equivalent to *conda config --describe > ~/.condarc*.

### Config Modifiers

**--get** Get a configuration value.

**--append** Add one configuration value to the end of a list key.

**--prepend, --add** Add one configuration value to the beginning of a list key.

**--set** Set a boolean or string key.

**--remove** **Remove a configuration value from a list key.**
This removes all instances of the value.

**--remove-key** Remove a configuration key (and all its values).

**--stdin** Apply configuration information given in yaml format piped through stdin.

See *conda config --describe* or https://conda.io/docs/config.html for details on all the options that can go in .condarc.

Examples:

Display all configuration values as calculated and compiled:

```
conda config --show
```

Display all identified configuration sources:

```
conda config --show-sources
```

Print the descriptions of all available configuration options to your command line:

```
conda config --describe
```

Print the description for the "channel_priority" configuration option to your command line:

```
conda config --describe channel_priority
```

Add the conda-canary channel:

```
conda config --add channels conda-canary
```

Set the output verbosity to level 3 (highest) for the current activate environment:

```
conda config --set verbosity 3 --env
```

Add the 'conda-forge' channel as a backup to 'defaults':

```
conda config --append channels conda-forge
```

### 4.3.5 conda create

Create a new conda environment from a list of specified packages.

To use the newly-created environment, use 'conda activate envname'. This command requires either the -n NAME or -p PREFIX option.

```
usage: conda create [-h] [--clone ENV] [-n ENVIRONMENT | -p PATH] [-c CHANNEL]
                    [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin] [--copy]
                    [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY] [-C]
                    [-k] [--offline] [--json] [--console CONSOLE] [-v] [-q]
                    [-d] [-y] [--download-only] [--show-channel-urls]
                    [--file FILE] [--no-default-packages] [--subdir SUBDIR]
                    [--solver SOLVER] [--dev]
                    [package_spec ...]
```

#### Positional Arguments

**package_spec** List of packages to install or update in the conda environment.

#### Named Arguments

**--clone** Create a new environment as a copy of an existing local environment.

**--file** Read package versions from the given file. Repeated file specifications can be passed (e.g. --file=file1 --file=file2).

**--dev** Use *sys.executable -m conda* in wrapper scripts instead of CONDA_EXE. This is mainly for use during tests where we test new conda sources against old Python versions.

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

#### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

**--subdir, --platform** Possible choices: emscripten-wasm32, wasi-wasm32, freebsd-64, linux-32, linux-64, linux-aarch64, linux-armv6l, linux-armv7l, linux-ppc64, linux-ppc64le, linux-riscv64, linux-s390x, osx-64, osx-arm64, win-32, win-64, win-arm64, zos-z

Use packages built for this platform. The new environment will be configured to remember this choice. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. Defaults to the current (native) platform.

### Solver Mode Modifiers

**--strict-channel-priority** Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

**--no-channel-priority** Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

**--no-deps** Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

**--only-deps** Only install dependencies.

**--no-pin** Ignore pinned file.

**--no-default-packages** Ignore create_default_packages in the .condarc file.

**--solver** Choose which solver backend to use.

### Package Linking and Install-time Options

**--copy** Install all packages using copies instead of hard- or soft-linking.

**--no-shortcuts** Don't install start menu shortcuts

**--shortcuts-only** Install shortcuts only for this package name. Can be used several times.

#### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

**--download-only** Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

**--show-channel-urls** Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

Examples:

Create an environment containing the package 'sqlite':

```
conda create -n myenv sqlite
```

Create an environment (env2) as a clone of an existing environment (env1):

```
conda create -n env2 --clone path/to/file/env1
```

### 4.3.6 conda deactivate

```
usage: conda deactivate [-h]
```

### 4.3.7 conda doctor

Display a health report for your environment.

```
usage: conda doctor [-v] [-h] [-n ENVIRONMENT | -p PATH]
```

#### Named Arguments

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### 4.3.8 conda env

```
usage: conda env [-h] command ...
```

#### Positional Arguments

**command** Possible choices: config, create, export, list, remove, update

#### conda env config

Configure a conda environment.

```
usage: conda env config [-h] {vars} ...
```

Examples:

```
conda env config vars list
conda env config --append channels conda-forge
```

### conda env config vars

Interact with environment variables associated with Conda environments.

```
usage: conda env config vars [-h] {list,set,unset} ...
```

Examples:

```
conda env config vars list -n my_env
conda env config vars set MY_VAR=something OTHER_THING=ohhhhya
conda env config vars unset MY_VAR
```

### conda env config vars list

List environment variables for a conda environment.

```
usage: conda env config vars list [-h] [-n ENVIRONMENT | -p PATH] [--json]
                                  [--console CONSOLE] [-v] [-q]
```

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

Example:

```
conda env config vars list -n my_env
```

### conda env config vars set

Set environment variables for a conda environment.

```
usage: conda env config vars set [-h] [-n ENVIRONMENT | -p PATH] [vars ...]
```

#### Positional Arguments

**vars** Environment variables to set in the form <KEY>=<VALUE> separated by spaces

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

Example:

```
conda env config vars set MY_VAR=weee
```

### conda env config vars unset

Unset environment variables for a conda environment.

```
usage: conda env config vars unset [-h] [-n ENVIRONMENT | -p PATH] [vars ...]
```

#### Positional Arguments

**vars** Environment variables to unset in the form <KEY> separated by spaces

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

Example:

```
conda env config vars unset MY_VAR
```

### conda env create

Create an environment based on an environment definition file.

If using an environment.yml file (the default), you can name the environment in the first line of the file with 'name: envname' or you can specify the environment name in the CLI command using the -n/--name argument. The name specified in the CLI will override the name specified in the environment.yml file.

Unless you are in the directory containing the environment definition file, use -f to specify the file path of the environment definition file you want to use.

```
usage: conda env create [-h] [-f FILE] [-n ENVIRONMENT | -p PATH] [-C] [-k]
                        [--offline]
                        [--environment-specifier ENVIRONMENT_SPECIFIER]
                        [--no-default-packages] [--json] [--console CONSOLE]
                        [-v] [-q] [-d] [-y] [--solver SOLVER]
                        [--subdir SUBDIR]
                        [remote_definition]
```

#### Positional Arguments

**remote_definition** *remote_definition* is deprecated and will be removed in 25.9. Use *conda env create --file=URL* instead.

#### Named Arguments

**-f, --file** Environment definition file (default: environment.yml)

**--environment-specifier, --env-spec** (EXPERIMENTAL) Specify the environment specifier plugin to use.

**--no-default-packages** Ignore create_default_packages in the .condarc file.

**--solver** Choose which solver backend to use.

**--subdir, --platform** Possible choices: emscripten-wasm32, wasi-wasm32, freebsd-64, linux-32, linux-64, linux-aarch64, linux-armv6l, linux-armv7l, linux-ppc64, linux-ppc64le, linux-riscv64, linux-s390x, osx-64, osx-arm64, win-32, win-64, win-arm64, zos-z

Use packages built for this platform. The new environment will be configured to remember this choice. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. Defaults to the current (native) platform.

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

```
conda env create
conda env create -n envname
conda env create folder/envname
conda env create -f /path/to/environment.yml
conda env create -f /path/to/requirements.txt -n envname
conda env create -f /path/to/requirements.txt -p /home/user/envname
```

## conda env export

Export a given environment

```
usage: conda env export [-h] [-c CHANNEL] [--override-channels]
                        [-n ENVIRONMENT | -p PATH] [-f FILE] [--no-builds]
                        [--ignore-channels] [--json] [--console CONSOLE] [-v]
                        [-q] [--from-history]
```

### Named Arguments

| | |
|---|---|
| **-c, --channel** | Additional channel to include in the export |
| **--override-channels** | Do not include .condarc channels |
| **-f, --file** | File name or path for the exported environment. Note: This will silently overwrite any existing file of the same name in the current directory. |
| **--no-builds** | Remove build specification from dependencies |
| **--ignore-channels** | Do not include channel names with package names. |
| **--from-history** | Build environment spec from explicit specs in history |

### Target Environment Specification

| | |
|---|---|
| **-n, --name** | Name of environment. |
| **-p, --prefix** | Full path to environment location (i.e. prefix). |

### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

Examples:

```
conda export
conda export --file FILE_NAME
```

## conda env list

An alias for *conda info --envs*. Lists all conda environments.

```
usage: conda env list [-h] [--json] [--console CONSOLE] [-v] [-q]
```

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

Examples:

```
conda env list
conda env list --json
```

## conda env remove

Remove an environment.

Removes a provided environment. You must deactivate the existing environment before you can remove it.

```
usage: conda env remove [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                        [--solver SOLVER] [--json] [--console CONSOLE] [-v]
                        [-q] [-d] [-y]
```

### Named Arguments

**--override-frozen** DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

**--solver** Choose which solver backend to use.

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

```
conda env remove --name FOO
conda env remove -n FOO
```

### conda env update

Update the current environment based on environment file.

```
usage: conda env update [-h] [--environment-specifier ENVIRONMENT_SPECIFIER]
                        [--override-frozen] [-n ENVIRONMENT | -p PATH]
                        [-f FILE] [--prune] [--json] [--console CONSOLE] [-v]
                        [-q] [--solver SOLVER]
                        [remote_definition]
```

#### Positional Arguments

**remote_definition** *remote_definition* is deprecated and will be removed in 25.9. Use *conda env update --file=URL* instead.

#### Named Arguments

**--environment-specifier, --env-spec** (EXPERIMENTAL) Specify the environment specifier plugin to use.

**--override-frozen** DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

**-f, --file** environment definition (default: environment.yml)

**--prune** remove installed packages not defined in environment.yml

**--solver** Choose which solver backend to use.

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

Examples:

```
conda env update
conda env update -n=foo
conda env update -f=/path/to/environment.yml
conda env update --name=foo --file=environment.yml
conda env update vader/deathstar
```

### 4.3.9 conda info

Display information about current conda install.

```
usage: conda info [-h] [--json] [--console CONSOLE] [-v] [-q] [-a] [--base]
                  [-e] [-s] [--unsafe-channels]
```

#### Named Arguments

| | |
|---|---|
| **-a, --all** | Show all information. |
| **--base** | Display base environment path. |
| **-e, --envs** | List all known conda environments. |
| **-s, --system** | List environment variables. |
| **--unsafe-channels** | Display list of channels with tokens exposed. |

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

### 4.3.10 conda init

Initialize conda for shell interaction.

```
usage: conda init [-h] [--all] [--condabin] [--user] [--no-user] [--system]
                  [--reverse] [--json] [--console CONSOLE] [-v] [-q] [-d]
                  [SHELLS ...]
```

#### Positional Arguments

**SHELLS** Possible choices: bash, fish, tcsh, xonsh, zsh, powershell

One or more shells to be initialized. If not given, the default value is 'bash' on unix and 'cmd.exe' & 'powershell' on Windows. Use the '--all' flag to initialize all shells. Available shells: ['bash', 'fish', 'powershell', 'tcsh', 'xonsh', 'zsh']

#### Named Arguments

**--all** Initialize all currently available shells.

**-d, --dry-run** Only display what would have been done.

#### setup type

**--condabin** Add 'condabin/' directory to PATH only. Does not install the shell function.

**--user** Initialize conda for the current user (default).

**--no-user** Don't initialize conda for the current user.

**--system** Initialize conda for all users on the system.

**--reverse** Undo effects of last conda init.

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

Key parts of conda's functionality require that it interact directly with the shell within which conda is being invoked. The *conda activate* and *conda deactivate* commands specifically are shell-level commands. That is, they affect the state (e.g. environment variables) of the shell context being interacted with. Other core commands, like *conda create* and *conda install*, also necessarily interact with the shell environment. They're therefore implemented in ways specific to each shell. Each shell must be configured to make use of them.

The --condabin option adds the "$CONDA_PREFIX/condabin" directory to the PATH environment variable. This directory only contains the conda executable and does not contain other executables from other packages installed in

the base environment. On most shells, a small snippet is added to the shell profile. For CMD, the PATH environment variable is modified directly in the registry.

This command makes changes to your system that are specific and customized for each shell. To see the specific files and locations on your system that will be affected before, use the '--dry-run' flag. To see the exact changes that are being or will be made to each location, use the '--verbose' flag.

IMPORTANT: After running *conda init*, most shells will need to be closed and restarted for changes to take effect.

### 4.3.11 conda install

Install a list of packages into a specified conda environment.

This command accepts a list of package specifications (e.g, bitarray=0.8) and installs a set of packages consistent with those specifications and compatible with the underlying environment. If full compatibility cannot be assured, an error is reported and the environment is not changed.

Conda attempts to install the newest versions of the requested packages. To accomplish this, it may update some packages that are already installed, or install additional packages. To prevent existing packages from updating, use the --freeze-installed option. This may force conda to install older versions of the requested packages, and it does not prevent additional dependency packages from being installed.

If you wish to skip dependency checking altogether, use the '--no-deps' option. This may result in an environment with incompatible packages, so this option must be used with great caution.

conda can also be called with a list of explicit conda package filenames (e.g. ./lxml-3.2.0-py27_0.tar.bz2). Using conda in this mode implies the --no-deps option, and should likewise be used with great caution. Explicit filenames and package specifications cannot be mixed in a single command.

```
usage: conda install [-h] [--revision REVISION] [--override-frozen]
                     [-n ENVIRONMENT | -p PATH] [-c CHANNEL] [--use-local]
                     [--override-channels] [--repodata-fn REPODATA_FNS]
                     [--experimental {jlap,lock}] [--no-lock]
                     [--repodata-use-zst | --no-repodata-use-zst]
                     [--strict-channel-priority] [--no-channel-priority]
                     [--no-deps | --only-deps] [--no-pin] [--copy]
                     [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY] [-C]
                     [-k] [--offline] [--json] [--console CONSOLE] [-v] [-q]
                     [-d] [-y] [--download-only] [--show-channel-urls]
                     [--file FILE] [--solver SOLVER] [--force-reinstall]
                     [--freeze-installed | --update-deps | -S | --update-all | --update-
↪specs]
                     [--clobber] [--dev]
                     [package_spec ...]
```

### Positional Arguments

**package_spec** List of packages to install or update in the conda environment.

### Named Arguments

**--revision** Revert to the specified REVISION.

**--override-frozen** DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

**--file** Read package versions from the given file. Repeated file specifications can be passed (e.g. --file=file1 --file=file2).

**--dev** Use *sys.executable -m conda* in wrapper scripts instead of CONDA_EXE. This is mainly for use during tests where we test new conda sources against old Python versions.

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Solver Mode Modifiers

**--strict-channel-priority** Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

**--no-channel-priority** Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

**--no-deps** Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

**--only-deps** Only install dependencies.

**--no-pin** Ignore pinned file.

**--solver** Choose which solver backend to use.

**--force-reinstall** Ensure that any user-requested package for the current operation is uninstalled and reinstalled, even if that package already exists in the environment.

**--freeze-installed, --no-update-deps** Do not update or change already-installed dependencies.

**--update-deps** Update dependencies that have available updates.

**-S, --satisfied-skip-solve** Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

**--update-all, --all** Update all installed packages in the environment.

**--update-specs** Update based on provided specifications.

### Package Linking and Install-time Options

**--copy** Install all packages using copies instead of hard- or soft-linking.

**--no-shortcuts** Don't install start menu shortcuts

**--shortcuts-only** Install shortcuts only for this package name. Can be used several times.

**--clobber** Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |
| **-d, --dry-run** | Only display what would have been done. |
| **-y, --yes** | Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc. |
| **--download-only** | Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix. |
| **--show-channel-urls** | Show channel urls. Overrides the value given by *conda config --show show_channel_urls*. |

Examples:

Install the package 'scipy' into the currently-active environment:

```
conda install scipy
```

Install a list of packages into an environment, myenv:

```
conda install -n myenv scipy curl wheel
```

Install a specific version of 'python' into an environment, myenv:

```
conda install -p path/to/myenv python=3.11
```

### 4.3.12 conda list

List installed packages in a conda environment.

```
usage: conda list [-h] [-n ENVIRONMENT | -p PATH] [--json] [--console CONSOLE]
                  [-v] [-q] [--show-channel-urls] [--fields LIST_FIELDS]
                  [--reverse] [-c] [-f] [--explicit] [--md5] [--sha256] [-e]
                  [-r] [--no-pip] [--auth]
                  [regex]
```

### Positional Arguments

**regex** List only packages matching this regular expression.

### Named Arguments

**--show-channel-urls** Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

**--fields** Comma-separated list of fields to print. Valid values: ['arch', 'build', 'build_number', 'channel', 'channel_name', 'constrains', 'depends', 'dist_str', 'features', 'fn', 'license', 'license_family', 'md5', 'name', 'noarch', 'package_type', 'requested_spec', 'sha256', 'size', 'subdir', 'timestamp', 'track_features', 'url', 'version'].

**--reverse** List installed packages in reverse order.

**-c, --canonical** Output canonical names of packages only.

**-f, --full-name** Only search for full names, i.e., ^<regex>$. --full-name NAME is identical to regex '^NAME$'.

**--explicit** List explicitly all installed conda packages with URL (output may be used by conda create --file).

**--md5** Add MD5 hashsum when using --explicit.

**--sha256** Add SHA256 hashsum when using --explicit.

**-e, --export** Output explicit, machine-readable requirement strings instead of human-readable lists of packages. This output may be used by conda create --file.

**-r, --revisions** List the revision history.

**--no-pip** Do not include pip-only installed packages.

**--auth** In explicit mode, leave authentication details in package URLs. They are removed by default otherwise.

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

Examples:

List all packages in the current environment:

```
conda list
```

List all packages in reverse order:

```
conda list --reverse
```

List all packages installed into the environment 'myenv':

```
conda list -n myenv
```

List all packages that begin with the letters "py", using regex:

```
conda list ^py
```

Save packages for future use:

```
conda list --export > package-list.txt
```

Reinstall packages from an export file:

```
conda create -n myenv --file package-list.txt
```

### 4.3.13 conda notices

Retrieve latest channel notifications.

Conda channel maintainers have the option of setting messages that users will see intermittently. Some of these notices are informational while others are messages concerning the stability of the channel.

```
usage: conda notices [-h] [-c CHANNEL] [--use-local] [--override-channels]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                     [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                     [--json] [--console CONSOLE] [-v] [-q]
```

#### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in

time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

Examples:

```
conda notices

conda notices -c defaults
```

### 4.3.14 conda package

Create low-level conda packages. (EXPERIMENTAL)

```
usage: conda package [-h] [-n ENVIRONMENT | -p PATH] [-w PATH [PATH ...]] [-r]
                     [-u] [--pkg-name PKG_NAME] [--pkg-version PKG_VERSION]
                     [--pkg-build PKG_BUILD]
```

#### Named Arguments

**-w, --which** Given some file's PATH, print which conda package the file came from.

**-r, --reset** Remove all untracked files and exit.

**-u, --untracked** Display all untracked files and exit.

**--pkg-name** Designate package name of the package being created.

**--pkg-version** Designate package version of the package being created.

**--pkg-build** Designate package build number of the package being created.

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### 4.3.15 conda repoquery

> **Note**
>
> The `conda repoquery` command is provided by the `conda-libmamba-solver`. See https://conda.github.io/conda-libmamba-solver/user-guide/subcommands/ for more information.

Advanced search for repodata.

```
usage: conda repoquery [-h] {whoneeds,depends,search} ...
```

#### Positional Arguments

**subcmd** Possible choices: whoneeds, depends, search

#### conda repoquery depends

> **Note**
>
> The `conda repoquery depends` command is provided by the `conda-libmamba-solver`. See https://conda.github.io/conda-libmamba-solver/user-guide/subcommands/ for more information.

```
usage: conda repoquery depends [-p PLATFORM] [--no-installed] [--pretty] [-a]
                               [--use-cache-only] [-t] [--recursive] [-h]
                               [-c CHANNEL] [--use-local]
                               [--override-channels]
                               [--repodata-fn REPODATA_FNS]
                               [--experimental {jlap,lock}] [--no-lock]
                               [--repodata-use-zst | --no-repodata-use-zst]
                               [-C] [-k] [--offline] [--json]
                               [--console CONSOLE] [-v] [-q]
                               specs [specs ...]
```

### Positional Arguments

**specs** The target package(s).

### Subcommand options

**-p, --platform** Platform/subdir to search packages for. Defaults to current platform.

**--no-installed** Do not search currently installed packages.

**--pretty** Prettier output with more details.

**-a, --all-channels** Look at all channels (for depends / whoneeds).

**--use-cache-only** Search in pkgs_dirs too

### Dependency options

**-t, --tree** Show dependencies in a tree-like format.

**--recursive** Show dependencies recursively.

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

## `conda repoquery search`

**Note**

The `conda repoquery search` command is provided by the `conda-libmamba-solver`. See https://conda.github.io/conda-libmamba-solver/user-guide/subcommands/ for more information.

```
usage: conda repoquery search [-p PLATFORM] [--no-installed] [--pretty] [-a]
                              [--use-cache-only] [-h] [-c CHANNEL]
                              [--use-local] [--override-channels]
                              [--repodata-fn REPODATA_FNS]
                              [--experimental {jlap,lock}] [--no-lock]
                              [--repodata-use-zst | --no-repodata-use-zst]
                              [-C] [-k] [--offline] [--json]
                              [--console CONSOLE] [-v] [-q]
                              specs [specs ...]
```

### Positional Arguments

**specs** The target package(s).

### Subcommand options

**-p, --platform** Platform/subdir to search packages for. Defaults to current platform.

**--no-installed** Do not search currently installed packages.

**--pretty** Prettier output with more details.

**-a, --all-channels** Look at all channels (for depends / whoneeds).

**--use-cache-only** Search in pkgs_dirs too

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

## `conda repoquery whoneeds`

> **Note**
>
> The `conda repoquery whoneeds` command is provided by the `conda-libmamba-solver`. See https://conda.github.io/conda-libmamba-solver/user-guide/subcommands/ for more information.

```
usage: conda repoquery whoneeds [-p PLATFORM] [--no-installed] [--pretty] [-a]
                                [--use-cache-only] [-t] [--recursive] [-h]
                                [-c CHANNEL] [--use-local]
                                [--override-channels]
                                [--repodata-fn REPODATA_FNS]
                                [--experimental {jlap,lock}] [--no-lock]
                                [--repodata-use-zst | --no-repodata-use-zst]
                                [-C] [-k] [--offline] [--json]
                                [--console CONSOLE] [-v] [-q]
                                specs [specs ...]
```

### Positional Arguments

| | |
|---|---|
| **specs** | The target package(s). |

### Subcommand options

| | |
|---|---|
| **-p, --platform** | Platform/subdir to search packages for. Defaults to current platform. |
| **--no-installed** | Do not search currently installed packages. |
| **--pretty** | Prettier output with more details. |
| **-a, --all-channels** | Look at all channels (for depends / whoneeds). |
| **--use-cache-only** | Search in pkgs_dirs too |

### Dependency options

**-t, --tree** Show dependencies in a tree-like format.

**--recursive** Show dependencies recursively.

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

### 4.3.16 conda remove

Remove a list of packages from a specified conda environment.

Use *--all* flag to remove all packages and the environment itself.

This command will also remove any package that depends on any of the specified packages as well---unless a replacement can be found without that dependency. If you wish to skip this dependency checking and remove just the requested packages, add the '--force' option. Note however that this may result in a broken environment, so use this with caution.

```
usage: conda remove [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                    [-c CHANNEL] [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--features] [--force-remove] [--no-pin] [--solver SOLVER]
                    [-C] [-k] [--offline] [--json] [--console CONSOLE] [-v]
                    [-q] [-d] [-y] [--all] [--keep-env] [--dev]
                    [package_name ...]
```

#### Positional Arguments

| | |
|---|---|
| **package_name** | Package names to remove from the environment. |

#### Named Arguments

| | |
|---|---|
| **--override-frozen** | DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen. |
| **--all** | Remove all packages, i.e., the entire environment. |
| **--keep-env** | Used with *--all*, delete all packages but keep the environment. |
| **--dev** | Use *sys.executable -m conda* in wrapper scripts instead of CONDA_EXE. This is mainly for use during tests where we test new conda sources against old Python versions. |

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Solver Mode Modifiers

**--features** Remove features (instead of packages).

**--force-remove, --force** Forces removal of a package without removing packages that depend on it. Using this option will usually leave your environment in a broken and inconsistent state.

**--no-pin** Ignore pinned package(s) that apply to the current operation. These pinned packages might come from a .condarc file or a file in <TARGET_ENVIRONMENT>/conda-meta/pinned.

**--solver** Choose which solver backend to use.

#### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

Remove the package 'scipy' from the currently-active environment:

```
conda remove scipy
```

Remove a list of packages from an environment 'myenv':

```
conda remove -n myenv scipy curl wheel
```

Remove all packages from environment *myenv* and the environment itself:

```
conda remove -n myenv --all
```

Remove all packages from the environment *myenv* but retain the environment:

```
conda remove -n myenv --all --keep-env
```

### 4.3.17 conda rename

Rename an existing environment.

This command renames a conda environment via its name (-n/--name) or its prefix (-p/--prefix).

The base environment and the currently-active environment cannot be renamed.

```
usage: conda rename [-h] [-n ENVIRONMENT | -p PATH] [--json]
                    [--console CONSOLE] [-v] [-q] [-d] [-y]
                    destination
```

#### Positional Arguments

**destination** New name for the conda environment.

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

Examples:

```
conda rename -n test123 test321

conda rename --name test123 test321

conda rename -p path/to/test123 test321

conda rename --prefix path/to/test123 test321
```

### 4.3.18 conda run

Run an executable in a conda environment.

```
usage: conda run [-h] [-n ENVIRONMENT | -p PATH] [-v] [--dev]
                 [--debug-wrapper-scripts] [--cwd CWD] [--no-capture-output]
                 ...
```

#### Positional Arguments

**executable_call** Executable name, with additional arguments to be passed to the executable on invocation.

#### Named Arguments

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**--dev** Sets *CONDA_EXE* to *python -m conda*, assuming the current working directory contains the root of conda development sources. This is mainly for use during tests where we test new conda sources against old Python versions.

**--debug-wrapper-scripts** When this is set, where implemented, the shell wrapper scriptswill use the echo command to print debugging information to stderr (standard error).

**--cwd** Current working directory for command to run in. Defaults to the user's current working directory if no directory is specified.

**--no-capture-output, --live-stream** Don't capture stdout/stderr (standard out/standard error).

#### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

Example:

```
$ conda create -y -n my-python-env python=3
$ conda run -n my-python-env python --version
```

### 4.3.19 conda search

Search for packages and display associated information using the MatchSpec format.

MatchSpec is a query language for conda packages.

```
usage: conda search [-h] [--envs] [-i] [--subdir SUBDIR]
                    [--skip-flexible-search] [-c CHANNEL] [--use-local]
                    [--override-channels] [--repodata-fn REPODATA_FNS]
                    [--experimental {jlap,lock}] [--no-lock]
                    [--repodata-use-zst | --no-repodata-use-zst] [-C] [-k]
                    [--offline] [--json] [--console CONSOLE] [-v] [-q]
```

### Named Arguments

**--envs** Search all of the current user's environments. If run as Administrator (on Windows) or UID 0 (on unix), search all known environments on the system.

**-i, --info** Provide detailed information about each package.

**--subdir, --platform** Search the given subdir. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. The default is to search the current platform.

**--skip-flexible-search** Do not perform flexible search if initial search fails.

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

| | |
|---|---|
| **--json** | Report all output as json. Suitable for using conda programmatically. |
| **--console** | Select the backend to use for normal output rendering. |
| **-v, --verbose** | Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging. |
| **-q, --quiet** | Do not display progress bar. |

Examples:

Search for a specific package named 'scikit-learn':

```
conda search scikit-learn
```

Search for packages containing 'scikit' in the package name:

```
conda search *scikit*
```

Note that your shell may expand '*' before handing the command over to conda. Therefore, it is sometimes necessary to use single or double quotes around the query:

```
conda search '*scikit'
conda search "*scikit*"
```

Search for packages for 64-bit Linux (by default, packages for your current platform are shown):

```
conda search numpy[subdir=linux-64]
```

Search for a specific version of a package:

```
conda search 'numpy>=1.12'
```

Search for a package on a specific channel:

```
conda search conda-forge::numpy
conda search 'numpy[channel=conda-forge, subdir=osx-64]'
```

### 4.3.20 conda update

Update conda packages to the latest compatible version.

This command accepts a list of package names and updates them to the latest versions that are compatible with all other packages in the environment.

Conda attempts to install the newest versions of the requested packages. To accomplish this, it may update some packages that are already installed, or install additional packages. To prevent existing packages from updating, use the --no-update-deps option. This may force conda to install older versions of the requested packages, and it does not prevent additional dependency packages from being installed.

```
usage: conda update [-h] [--override-frozen] [-n ENVIRONMENT | -p PATH]
                    [-c CHANNEL] [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--no-lock] [--repodata-use-zst | --no-repodata-use-zst]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin] [--copy]
                    [--no-shortcuts] [--shortcuts-only SHORTCUTS_ONLY] [-C]
                    [-k] [--offline] [--json] [--console CONSOLE] [-v] [-q]
                    [-d] [-y] [--download-only] [--show-channel-urls]
                    [--file FILE] [--solver SOLVER] [--force-reinstall]
                    [--freeze-installed | --update-deps | -S | --update-all | --update-
↪specs]
                    [--clobber]
                    [package_spec ...]
```

### Positional Arguments

**package_spec** List of packages to install or update in the conda environment.

### Named Arguments

**--override-frozen** DANGEROUS. Use at your own risk. Ignore protections if the environment is frozen.

**--file** Read package versions from the given file. Repeated file specifications can be passed (e.g. --file=file1 --file=file2).

### Target Environment Specification

**-n, --name** Name of environment.

**-p, --prefix** Full path to environment location (i.e. prefix).

### Channel Customization

**-c, --channel** Additional channel to search for packages. These are URLs searched in the order they are given (including local directories using the 'file://' syntax or simply a path like '/home/conda/mychan' or '../mychan'). Then, the defaults or channels from .condarc are searched (unless --override-channels is given). You can use 'defaults' to get the default packages for conda. You can also use any name and the .condarc channel_alias value will be prepended. The default channel_alias is https://conda.anaconda.org/.

**--use-local** Use locally built packages. Identical to '-c local'.

**--override-channels** Do not search default or .condarc channels. Requires --channel.

**--repodata-fn** Specify file name of repodata on the remote server where your channels are configured or within local backups. Conda will try whatever you specify, but will ultimately fall back to repodata.json if your specs are not satisfiable with what you specify here. This is used to employ repodata that is smaller and reduced in time scope. You may pass this flag more than once. Leftmost entries are tried

first, and the fallback to repodata.json is added for you automatically. For more information, see conda config --describe repodata_fns.

**--experimental** Possible choices: jlap, lock

jlap: Download incremental package index data from repodata.jlap; implies 'lock'. lock: use locking when reading, updating index (repodata.json) cache. Now enabled.

**--no-lock** Disable locking when reading, updating index (repodata.json) cache.

**--repodata-use-zst, --no-repodata-use-zst** Check for/do not check for repodata.json.zst. Enabled by default.

### Solver Mode Modifiers

**--strict-channel-priority** Packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.

**--no-channel-priority** Package version takes precedence over channel priority. Overrides the value given by *conda config --show channel_priority*.

**--no-deps** Do not install, update, remove, or change dependencies. This WILL lead to broken environments and inconsistent behavior. Use at your own risk.

**--only-deps** Only install dependencies.

**--no-pin** Ignore pinned file.

**--solver** Choose which solver backend to use.

**--force-reinstall** Ensure that any user-requested package for the current operation is uninstalled and reinstalled, even if that package already exists in the environment.

**--freeze-installed, --no-update-deps** Do not update or change already-installed dependencies.

**--update-deps** Update dependencies that have available updates.

**-S, --satisfied-skip-solve** Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by the 'aggressive_update_packages' config setting. Use 'conda config --describe aggressive_update_packages' to view your setting. --satisfied-skip-solve is similar to the default behavior of 'pip install'.

**--update-all, --all** Update all installed packages in the environment.

**--update-specs** Update based on provided specifications.

### Package Linking and Install-time Options

**--copy** Install all packages using copies instead of hard- or soft-linking.

**--no-shortcuts** Don't install start menu shortcuts

**--shortcuts-only** Install shortcuts only for this package name. Can be used several times.

**--clobber** Allow clobbering of overlapping file paths within packages, and suppress related warnings.

#### Networking Options

**-C, --use-index-cache** Use cache of channel index files, even if it has expired. This is useful if you don't want conda to check whether a new version of the repodata file exists, which will save bandwidth.

**-k, --insecure** Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

**--offline** Offline mode. Don't connect to the Internet.

#### Output, Prompt, and Flow Control Options

**--json** Report all output as json. Suitable for using conda programmatically.

**--console** Select the backend to use for normal output rendering.

**-v, --verbose** Can be used multiple times. Once for detailed output, twice for INFO logging, thrice for DEBUG logging, four times for TRACE logging.

**-q, --quiet** Do not display progress bar.

**-d, --dry-run** Only display what would have been done.

**-y, --yes** Sets any confirmation values to 'yes' automatically. Users will not be asked to confirm any adding, deleting, backups, etc.

**--download-only** Solve an environment and ensure package caches are populated, but exit prior to unlinking and linking packages into the prefix.

**--show-channel-urls** Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

Examples:

```
conda update -n myenv scipy
```

### 4.3.21 Conda vs. pip vs. virtualenv commands

If you have used pip and virtualenv in the past, you can use conda to perform all of the same operations. Pip is a package manager and virtualenv is an environment manager. conda is both.

Scroll to the right to see the entire table.

| Task | Conda package and environment manager command | Pip package manager command | Virtualenv environment manager command |
|---|---|---|---|
| Install a package | `conda install $PACKAGE_NAME` | `pip install $PACKAGE_NAME` | X |
| Update a package | `conda update --name $ENVIRONMENT_NAME $PACKAGE_NAME` | `pip install --upgrade $PACKAGE_NAME` | X |
| Update package manager | `conda update conda` | Linux/macOS: `pip install -U pip` Win: `python -m pip install -U pip` | X |
| Uninstall a package | `conda remove --name $ENVIRONMENT_NAME $PACKAGE_NAME` | `pip uninstall $PACKAGE_NAME` | X |
| Create an environment | `conda create --name $ENVIRONMENT_NAME python` | X | `cd $ENV_BASE_DIR; virtualenv $ENVIRONMENT_NAME` |
| Activate an environment | `conda activate $ENVIRONMENT_NAME*` | X | `source $ENV_BASE_DIR/ $ENVIRONMENT_NAME/bin/ activate` |
| Deactivate an environment | `conda deactivate` | X | `deactivate` |
| Search available packages | `conda search $SEARCH_TERM` | `pip search $SEARCH_TERM` | X |
| Install package from specific source | `conda install --channel $URL $PACKAGE_NAME` | `pip install --index-url $URL $PACKAGE_NAME` | X |
| List installed packages | `conda list --name $ENVIRONMENT_NAME` | `pip list` | X |
| Create requirements file | `conda list --export` | `pip freeze` | X |
| List all environments | `conda info --envs` | X | Install virtualenv wrapper, then `lsvirtualenv` |
| Install other package manager | `conda install pip` | `pip install conda` | X |
| Install Python | `conda install python=x.x` | X | X |
| Update Python | `conda update python*` | X | X |

* `conda activate` only works on conda 4.6 and later versions. For conda versions prior to 4.6, type:

  - Windows: `activate`
  - Linux and macOS: `source activate`

* `conda update python` updates to the most recent in the series, so any Python 2.x would update to the latest 2.x and any Python 3.x to the latest 3.x.

## 4.4 Release notes

This information is drawn from the GitHub conda project changelog: https://github.com/conda/conda/blob/main/CHANGELOG.md

### 4.4.1 25.5.1 (2025-06-05)

#### Bug fixes

- Allow `conda config` commands to process aliased settings names by forwarding them to the canonical name (e.g. `self_update` -> `auto_update_conda`). (#14899 via #14898)

#### Docs

- Mark environment spec hooks and types as experimental (#14900)

#### Contributors

- @jaimergp
- @soapy1
- @travishathaway

### 4.4.2 25.5.0 (2025-05-21)

#### Enhancements

- Add the ability to define pre- and post-transaction hooks. (#14754)
- Add support for showing and manipulating settings registered via the settings plugin hook to `conda config`. (#13661 via #14510)
- Raise `conda.exceptions.OfflineError` instead of generic `RuntimeError` when `--offline` is used for better error handling. (#14664 via #14665)
- Add `context.default_activation_env` setting, which allows users to customize which environment should get auto activated on initialization (if `context.auto_activate` is enabled), as well as `conda activate` calls without arguments. Defaults to `base`. (#14666)
- Add `conda init --condabin` to only add the `$CONDA_PREFIX/condabin` directory to the `PATH` environment variable instead of installing a shell function. (#14703)
- Add `environment_specifiers` plugin hooks to allow the creation of plugins to read different file formats and sources. (#14710)

- Extend `PrefixData` API with two constructors (`from_name()`, `from_context()`), a new property (`name`), several boolean-returning methods (`exists()`, `is_environment()`, `is_base()`), and a few exception-raising methods (`assert_exists()`, `assert_environment()`, `assert_writable()`, `validate_path()`, `validate_name()`). An additional method `set_nonadmin()` allows to plant the `.nonadmin` marker on Windows. (#14747 via #14750)
- Add support for frozen environment markers (CEP 22). (#14746 via #14766)
- Add a plugin hook for `PrefixData` loaders and use it to wrap the existing PyPI interoperability features. These are now present in `conda.plugins.prefix_data_loaders.pypi`. (#14780)
- Add a `--fields` flag to `conda list` that allows to customize the content of the `human` output format. (#14781)
- Add debug logs to get visibility into environment spec plugin selection. (#14815)
- Add `conda.models.records.SolvedRecord` class. It is a `PackageRecord` subclass augmented with the `requested_spec` field originally found in `PrefixRecord`. (#14821)

### Bug fixes

- Cleanup references to `DepsModifier.UPDATE_SPECS`. (#14807)
- Report package variant installation as "REVISED" as opposed to "DOWNGRADED". (#13797 via #14727)
- Add a validation step to check if the target environment is writable when running the `install` or `update` commands. (#12561 via #14668)
- Fix cache key in `PrefixData`. It will now consider `interoperability`. (#14750)
- Do not cache `PrefixData` records across successive `conda list` invocations. (#14750)
- Fix `environment.yaml` spec plugin to only handle yaml files (files with `.yaml` or `.yml` file extensions). (#14823)
- Fix `conda info` rendering to display components in a consistent order. (#14829 via #14831)

### Deprecations

- Mark `conda.base.context.context.auto_activate_base` as pending deprecation, to be removed in 26.3. Use `conda.base.context.context.auto_activate` instead. (#14666)
- Mark `conda.base.context.Context.pip_interop_enabled` as pending deprecation, to be removed in 26.3. Use `conda.base.context.Context.prefix_data_interoperability`. instead. (#14750)
- Mark `conda.base.context._first_writable_envs_dir` as pending deprecation, to be removed in 26.3. Use `conda.gateways.disk.create.first_writable_envs_dir()` and `PrefixData.from_name()` instead. (#14750)
- Mark `conda.base.context.validate_prefix_name` as pending deprecation, to be removed in 26.3. Use `PrefixData.validate_name()` and `PrefixData.from_name()` instead. (#14750)
- Mark `conda.cli.common.validate_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.assert_environment()` instead. (#14750)
- Mark `conda.cli.common.validate_prefix_is_writable` as pending deprecation, to be removed in 26.3. Use `PrefixData.assert_writable()` instead. (#14750)
- Mark `conda.cli.install.check_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()`, `PrefixData.validate_path()`, `PrefixData.validate_name()` instead. (#14750)
- Mark `conda.cli.install.print_activate` as pending deprecation, to be removed in 26.3. Use `conda.cli.common.print_activate` instead. (#14670)

- Mark `conda.cli.install.validate_new_prefix` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()` and `PrefixData.validate_path()` instead. (#14750)
- Mark `conda.cli.install.validate_prefix_exists` as pending deprecation, to be removed in 26.3. Use `PrefixData.exists()` instead. (#14750)
- Mark `conda.cli.main_info.get_info_components` as pending deprecation, to be removed in 26.3. Use `conda.cli.main_info.iter_info_components` instead. (#14837)
- Mark `conda.cli.main_rename.check_protected_dirs` as pending deprecation, to be removed in 26.3. Use `PrefixData.validate_path()` instead. (#14750)
- Mark `conda.cli.main_rename.validate_src` as pending deprecation. to be removed in 26.3. Use `PrefixData.validate_path()` and `PrefixData.validate_name()` instead. (#14750)
- Mark `conda.common.pkg_formats.python` module as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format` instead. (#14798)
- Mark `conda.common.pkg_formats` subpackage as pending deprecation, to be removed in 26.3. Use the `prefix_data_loaders` plugin hook instead. (#14798)
- Mark `conda.common.url.hex_octal_to_int` as pending deprecation, to be removed in 26.3. Use `int(..., 16)` instead. (#14750)
- Mark `conda.core.link.PrefixActionGroup` as pending deprecation, to be removed in 26.3. Use `conda.core.link.PrefixActions` instead. (#14754)
- Mark `conda.core.link.PrefixActions` as pending deprecated, to be removed in 26.3. Use `conda.core.link.PrefixActionGroup` instead. (#14754)
- Mark `conda.core.path_actions._Action` as pending deprecation, to be removed in 26.3. Use `conda.core.path_actions.Action` instead. (#14754)
- Mark `conda.core.prefix_data.get_python_version_for_prefix()` as pending deprecation, to be removed in 26.3. Use `conda.core.prefix_data.PrefixData(prefix).get("python").version` instead. (#14750)
- Mark `conda.core.prefix_data.PrefixData` keyword argument `pip_interop_enabled` as pending deprecation, to be removed in 26.3. Use the `interoperability` keyword argument instead. (#14750)
- Mark `conda.core.prefix_data.PrefixData._load_site_packages()` as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.load_site_packages()` instead. (#14750)
- Mark `conda.core.prefix_data.PrefixData._python_pkg_record` as pending deprecation, to be removed in 26.3. Use `PrefixData.get("python")` instead. (#14750)
- Mark `conda.core.prefix_data.python_record_for_prefix()` as pending deprecation, to be removed in 26.3. Use `conda.core.prefix_data.PrefixData(prefix).get("python")` instead. (#14750)
- Mark `conda.gateways.disk.read.read_python_record` as pending deprecation, to be removed in 26.3. Use `conda.plugins.prefix_data_loaders.pypi.pkg_format.read_python_record` instead. (#14798)
- Mark `conda.gateways.disk.test.is_conda_environment` as pending deprecation, to be removed in 26.3. Use `PrefixData.is_environment()` instead. (#14750)
- Mark `conda.gateways.disk.test.touch_nonadmin` as pending deprecation, to be removed in 26.3. Use `PrefixData.set_nonadmin()` instead. (#14750)
- Mark `conda.models.records.PackageRecord.schannel` as pending deprecation, to be removed in 26.3. Use `conda.models.records.PackageRecord.channel_name` instead. (#14781)

- Mark `conda.trust.signature_verification` and `conda.trust.constants` as pending deprecation, to be removed in 26.3. The functionality will be moved to `conda-content-trust` and `conda-anaconda-trust-root` packages respectively. (#14849)
- Mark `conda info --license` as pending deprecation, to be removed in 26.3. (#14831)
- Mark `conda info --root` as pending deprecation, to be removed in 26.3. Use `conda info --base` instead. (#14831)

### Docs

- Add type hints and docstrings to `conda.core.portability`. (#13820)
- Add type hints and docstrings to `conda.core.subdir_data`. (#13821)
- Add type hints to `conda.base.constants`. (#13480)
- Add type hints to `conda.base.context`. (#14776)
- Add type hints to `conda.__init__`, `conda.exceptions` and `conda.exception_handler`. (#14776)
- Add type hints to `conda.core.prefix_data`. (#14779)
- Add examples to environment specs plugin docs. (#14814)
- Add type hints to `conda.models.channel.Channel`. (#14817)
- Update the conda cheatsheet to 25.3.1 version. (#14830)

### Other

- Refactor `--repodata-fn` iteration in `conda.cli.install` and other smaller cleanups. (#14670)
- Refactor command line argument validations from `conda.cli.install.install` to the relevant `cli` modules. (#14742)
- Refactor `conda create --clone` logic into a separate function. (#14743)
- Use `dict.fromkeys` for sequence deduplication, instead of `IndexedSet()`. (#14777)

### Contributors

- @conda-bot
- @faithrider
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @peytondmurray
- @samhaese
- @soapy1
- @travishathaway
- @dependabot[bot]

- @pre-commit-ci[bot]

#### New Contributors

- @faithrider made their first contribution in https://github.com/conda/conda/pull/13820
- @peytondmurray made their first contribution in https://github.com/conda/conda/pull/14754

### 4.4.3 25.3.1 (2025-04-03)

#### Bug fixes

- Restore CMD prompt modifier. (#14711 via #14713)

#### Contributors

- @kenodegard

### 4.4.4 25.3.0 (2025-03-17)

#### Enhancements

- Add support for Python 3.13. (#14584)
- Refactor CMD's activation process to use a static shell script instead of a dynamic temporary script. This resolves using `conda activate` with script execution restrictions when Windows AppLocker is enabled. (#13610 via #14607)

#### Bug fixes

- Fix variable deletion in xonsh. (#14543 via #14550)
- Make `conda.gateways.logging.TokenURLFilter.TOKEN_REPLACE` a staticmethod for future Python compatibility. (#14590)
- Allow user to specify `--name` or `--prefix` as part of `conda config` command. (#12137 via #14648)

#### Deprecations

- Postpone `conda.base.context.Context.conda_exe` deprecation to `26.3`. (#14647)
- Postpone `conda.base.context.Context.restore_free_channel` deprecation to `25.9`. (#14647)
- Postpone implicit `defaults` multichannel deprecation to `25.9`. Users relying on this behavior are encouraged to run `conda config --add channels defaults`. (#14178 via #14662)
- Postpone loading subcommands from executables deprecation to `conda 26.3`. (#14644)
- Mark `conda {create,install,update} -f` as pending deprecation. Use `conda {create,install,update} --file` instead. (#14446)
- Mark `conda._vendor` as pending deprecation. Conda no longer vendors dependencies. (#14562)
- Mark `conda.common.io._logger_lock` as pending deprecation. Use `logging._lock` instead. (#14584)

- Mark `conda.exports.plan` as pending deprecation. (#14631)
- Mark `conda.plan.Channel` as pending deprecation. Use `conda.models.channel.Channel` instead. (#14603)
- Mark `conda.plan.context` as pending deprecation. Use `conda.base.context.context` instead. (#14603)
- Mark `conda.plan.dashlist` as pending deprecation. Use `conda.common.io.dashlist` instead. (#14603)
- Mark `conda.plan.defaultdict` as pending deprecation. Use builtin `collections.defaultdict` instead. (#14603)
- Mark `conda.plan.DEFAULTS_CHANNEL_NAME` as pending deprecation. Use `conda.base.constants.DEFAULTS_CHANNEL_NAME` instead. (#14603)
- Mark `conda.plan.Dist` as pending deprecation. Use `conda.models.dist.Dist` instead. (#14603)
- Mark `conda.plan.env_vars` as pending deprecation. Use `conda.common.io.env_vars` instead. (#14603)
- Mark `conda.plan.FETCH` as pending deprecation. Use `conda.instructions.FETCH` instead. (#14603)
- Mark `conda.plan.groupby_to_dict` as pending deprecation. Use `conda.common.iterators.groupby_to_dict` instead. (#14603)
- Mark `conda.plan.human_bytes` as pending deprecation. Use `conda.utils.human_bytes` instead. (#14603)
- Mark `conda.plan.IndexedSet` as pending deprecation. Use `boltons.setutils.IndexedSet` instead. (#14603)
- Mark `conda.plan.LAST_CHANNEL_URLS` as pending deprecation. Use `conda.core.index.LAST_CHANNEL_URLS` instead. (#14603)
- Mark `conda.plan.LINK` as pending deprecation. Use `conda.instructions.LINK` instead. (#14603)
- Mark `conda.plan.LinkType` as pending deprecation. Use `conda.models.enums.LinkType` instead. (#14603)
- Mark `conda.plan.log` as pending deprecation. Use builtin `logging` instead. (#14603)
- Mark `conda.plan.MatchSpec` as pending deprecation. Use `conda.models.match_spec.MatchSpec` instead. (#14603)
- Mark `conda.plan.normalized_version` as pending deprecation. Use `conda.models.version.normalized_version` instead. (#14603)
- Mark `conda.plan.PackageRecord` as pending deprecation. Use `conda.models.records.PackageRecord` instead. (#14603)
- Mark `conda.plan.PrefixSetup` as pending deprecation. Use `conda.core.link.PrefixSetup` instead. (#14603)
- Mark `conda.plan.prioritize_channels` as pending deprecation. Use `conda.models.channel.prioritize_channels` instead. (#14603)
- Mark `conda.plan.reset_context` as pending deprecation. Use `conda.base.context.reset_context` instead. (#14603)
- Mark `conda.plan.SYMLINK_CONDA` as pending deprecation. Use `conda.instructions.SYMLINK_CONDA` instead. (#14603)
- Mark `conda.plan.sys` as pending deprecation. Use builtin `sys` instead. (#14603)
- Mark `conda.plan.time_recorder` as pending deprecation. Use `conda.common.io.time_recorder` instead. (#14603)

- Mark `conda.plan.TRACE` as pending deprecation. Use `conda.common.constants.TRACE` instead. (#14603)
- Mark `conda.plan.UNKNOWN_CHANNEL` as pending deprecation. Use `conda.base.constants.UNKNOWN_CHANNEL` instead. (#14603)
- Mark `conda.plan.UNLINK` as pending deprecation. Use `conda.instructions.UNLINK` instead. (#14603)
- Mark `conda.plan.UnlinkLinkTransaction` as pending deprecation. Use `conda.core.link.UnlinkLinkTransaction` instead. (#14603)
- Mark `conda.plan` as pending deprecation. (#14603)
- Mark `conda.testing.fixtures.session_capsys` as pending deprecation. (#14575)
- Mark `conda_env.cli` as pending deprecation. (#14564)
- Mark `conda_env.installers` as pending deprecation. (#14564)
- Mark `conda_env` as pending deprecation. (#14564)
- Remove `conda create --mkdir`. Redundant argument. (#14644)
- Remove `conda install --mkdir`. Use `conda create` instead. (#14644)
- Remove `conda rename --force`. Use `conda rename --yes` instead. (#14644)
- Remove `conda._vendor.frozendict`. Use `frozendict` instead. (#14562, #14582)
- Remove `conda.activate._Activator._get_path_dirs(extra_library_bin)`. (#14563)
- Remove `conda.activate._Activator.add_export_unset_vars`. Use `conda.activate._Activator.get_export_unset_vars` instead. (#14563)
- Remove `conda.activate._Activator.get_scripts_export_unset_vars`. Use `get_scripts_export_unset_vars` helper function in `test_activate.py` instead. (#14563)
- Remove `conda.activate.JSONFormatMixin.get_scripts_export_unset_vars`. Use `conda.activate._Activator.get_export_unset_vars` instead. (#14563)
- Remove `conda.auxlib.collection.make_immutable`. Use `frozendict.deepfreeze` instead. (#14586)
- Remove `conda.cli.main_env.configure_parser(sub_parsers=None)`. (#14600)
- Remove `conda.cli.main_env_list.execute`. Use `conda.cli.main_info.execute(envs=True)` instead. (#14604)
- Remove `conda.cli.main_info.get_info_dict(system)`. (#14604)
- Remove `conda.common.compat.encode_arguments`. (#14604)
- Remove `conda.env.specs.detect(remote_definition)`. (#14604)
- Remove `conda.instructions.PREFIX`. (#14604)
- Remove `conda.plan._get_best_prec_match`. Use `conda.misc._get_best_prec_match` instead. (#14603)
- Remove `conda.plan._handle_menuinst`. (#14603)
- Remove `conda.plan._inject_UNLINKLINKTRANSACTION`. (#14603)
- Remove `conda.plan._plan_from_actions`. (#14603)
- Remove `conda.plan._update_old_plan`. (#14603)
- Remove `conda.plan.add_defaults_to_specs`. (#14603)
- Remove `conda.plan.add_unlink`. (#14603)

- Remove `conda.plan.display_actions`. (#14603)
- Remove `conda.plan.execute_actions`. (#14603)
- Remove `conda.plan.execute_plan`. (#14603)
- Remove `conda.plan.execute_plan`. (#14603)
- Remove `conda.plan.get_blank_actions`. (#14603)
- Remove `conda.plan.install_actions`. (#14603)
- Remove `conda.plan.print_dists`. (#14603)
- Remove `conda.plan.revert_actions`. Use `conda.cli.install.revert_actions` instead. (#14603)
- Remove `conda.plan` as an entrypoint. (#14603)
- Remove `conda.testing.integration._get_temp_prefix`. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.create_temp_location`. Use `tmp_path` or `conda.testing.fixtures.path_factory` fixtures instead. (#14604)
- Remove `conda.testing.integration.FORCE_temp_prefix`. Use `tmp_path`, `conda.testing.fixtures.path_factory`, or `conda.testing.fixtures.tmp_env` fixtures instead. (#14604)
- Remove `conda.testing.integration.make_temp_channel`. Use `conda.testing.fixtures.tmp_channel` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_env`. Use `conda.testing.fixtures.tmp_env` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_package_cache`. Use `conda.testing.fixtures.tmp_pkgs_dir` fixture instead. (#14604)
- Remove `conda.testing.integration.make_temp_prefix`. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.run_command`. Use `conda.testing.conda_cli` instead. (#14645)
- Remove `conda.testing.integration.running_a_python_capable_of_unicode_subprocessing`. (#14669)
- Remove `conda.testing.integration.set_tmpdir` fixture. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#14645)
- Remove `conda.testing.integration.tempdir`. Use `tmp_path` or `conda.testing.fixtures.path_factory` fixtures instead. (#14604)
- Remove `conda_env.cli.common`. Use `conda.env.env` instead. (#14564)
- Remove `conda_env.cli.main_config`. Use `conda.cli.main_env_config` instead. (#14564)
- Remove `conda_env.cli.main_create`. Use `conda.cli.main_env_create` instead. (#14564)
- Remove `conda_env.cli.main_export`. Use `conda.cli.main_export` instead. (#14564, #14601)
- Remove `conda_env.cli.main_list`. Use `conda.cli.main_env_list` instead. (#14564)
- Remove `conda_env.cli.main_remove`. Use `conda.cli.main_env_remove` instead. (#14564)
- Remove `conda_env.cli.main_update`. Use `conda.cli.main_env_update` instead. (#14564)
- Remove `conda_env.cli.main_vars`. Use `conda.cli.main_env_vars` instead. (#14564)
- Remove `conda_env.env`. Use `conda.env.env` instead. (#14564)

- Remove `conda_env.installers.base`. Use `conda.env.installers.base` instead. (#14564)
- Remove `conda_env.installers.conda`. Use `conda.env.installers.conda` instead. (#14564)
- Remove `conda_env.installers.pip`. Use `conda.env.installers.pip` instead. (#14564)
- Remove `conda_env.pip_util`. Use `conda.env.pip_util` instead. (#14564)
- Remove `conda_env.specs.binstar`. Use `conda.env.specs.binstar` instead. (#14564)
- Remove `conda_env.specs.requirements`. Use `conda.env.specs.requirements` instead. (#14564)
- Remove `conda_env.specs.yaml_file`. Use `conda.env.specs.yaml_file` instead. (#14564)
- Remove `conda_env.specs`. Use `conda.env.specs` instead. (#14564)
- Remove `logging.Logger.trace` monkeypatch. Use `logging.getLogger(__name__)(conda.common.constants.TRACE, ...)` instead. (#14647)

#### Contributors

- @v2thegreat made their first contribution in https://github.com/conda/conda/pull/14581
- @conda-bot
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @soapy1
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

### 4.4.5 25.1.1 (2025-01-28)

#### Bug fixes

- Fix PowerShell activation/deactivation to properly unset `$env:_CE_M` and `$env:_CE_CONDA` environment variables. (#14292 via #14517)

#### Contributors

- @travishathaway

### 4.4.6 25.1.0 (2025-01-17)

#### Enhancements

- Set `__win` version and enable `CONDA_OVERRIDE_WIN` usage. (#14443 via #14450)

#### Bug fixes

- Merge overlapping glob build specs instead of raising `Incompatible component merge`. (#11612)
- Fix a bug when invalid values are being passed to `conda shell.posix` command (#14398)
- Sort suggested subcommands when an incorrect subcommand is provided in the CLI. (#13332 via 14402)
- Fix a bug where the setting `denylist_channels` was not being recognized in certain cases. (#14405)
- Do not use native platform version to set `__osx` or `__linux` version if the underlying OS is not macOS or Linux, respectively. (#14448 via #14449)
- Report real macOS version (11+ instead of 10.16) even if the Python interpreter was linked against SDK 10.15 or earlier. This applies to the `__osx` virtual package and the user agent info. (#13178, #13832 via #14449)
- Accept %-encoded URLs as a valid `MatchSpec`. (#14481)
- Retry failed downloads one time on `ChecksumMismatchError` as caused by bad partial downloads. Use `r+b` or `w+b` instead of "append" mode. Improve test coverage. (#13488)

#### Docs

- Use the correct parameter name, `handler`, in the CondaAuthHandler example (#14428).

#### Other

- Require `conda-libmamba-solver >=24.11.0` for `libmamba 2.x` compatibility. (#11612)
- Added new `PYTHONPATH` autoused fixture in `conda.testing` to ensure development `conda` is used across all tests. The fixture doesn't apply in `PYTHONPATH` is already set in the environment. (#14475)

#### Contributors

- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @ForgottenProgramme
- @minrk
- @travishathaway

- @dependabot[bot]
- @pre-commit-ci[bot]

### 4.4.7 24.11.3 (2025-01-06)

#### Bug fixes

- Fix recursion error introduced in `__conda_reactivate` deprecation message. (#14468)

#### Contributors

- @jaimergp

### 4.4.8 24.11.2 (2024-12-19)

#### Bug fixes

- Restore `__conda_reactivate` shell command removed in 24.11.0. (#14455)

#### Deprecations

- Mark `__conda_reactivate` as deprecated. Use `__conda_activate reactivate` instead. (#14455)

#### Contributors

- @kenodegard

### 4.4.9 24.11.1 (2024-12-10)

#### Bug fixes

- Fix a bug where the setting `denylist_channels` was not being recognized in certain cases. (#14405)

#### Contributors

- @travishathaway

### 4.4.10 24.11.0 (2024-11-18)

#### Enhancements

- Adds a new plugin hook for reporter backends. (#13868)
- Add support for CEP-17 that allows specifying the path to the site-packages directory of the the Python package via the `repodata.json`. (#14053 via #14256)
- Adds progress bar support for reporter backends plugin hook. (#14083)

- Adds support for defining spinners for the reporter backends plugin hook. (#14206)
- Adds support for confirmation functions for reporter backends plugin hook. (#14244)
- Add new plugin hooks (`conda_session_headers` and `conda_request_headers`) to add headers to outgoing HTTP requests. (#14325, #14382)

### Bug fixes

- Do not retry solves twice in failed `conda env` runs. (#13784)
- Remove CreateNonAdminAction to prevent conda remove from deleting `.nonadmin` files. (#14271)
- Do not map Python distribution names to conda names in `PrefixData(pip_interop_enabled=True)`. (#14310 via #14317)
- Fix output writing for `conda export --json --file`. (#14316 via #14323)
- Update `deprecated.action()` function to account for positional arguments that have no value specified. (#14355 via #14359)
- Fix continuous integration upload of coverage files. (#14375)

### Deprecations

- Remove `__conda_reactivate` shell function in favor of `__conda_activate reactivate`. (#14277)
- Mark `conda.misc.rel_path` as pending deprecation. (#14338)
- Require Python 3.9 or greater. (#14201 via #14368)

### Contributors

- @beeankha
- @conda-bot
- @dholth
- @duncanmmacleod
- @jaimergp
- @jezdez
- @jjhelmus
- @kathatherine
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @marcoesters
- @muffato made their first contribution in https://github.com/conda/conda/pull/14342
- @nilskch made their first contribution in https://github.com/conda/conda/pull/14214
- @travishathaway
- @dependabot[bot]

- @pre-commit-ci[bot]

### 4.4.11 24.9.2 (2024-10-16)

#### Bug fixes

- Refine the deprecation warning to the prepopulation of the channel list with "defaults". (#14305)
- Add protected directories guardrail to `conda create` command. (#14282, #14315)

#### Contributors

- @beeankha
- @jezdez
- @travishathaway

### 4.4.12 24.9.1 (2024-10-01)

#### Bug fixes

- Tweak pending deprecation warning for the upcoming changes in channel defaults. (#14287 via #14288)

#### Deprecations

- Remove deprecated testing entrypoint `shell/bin/conda`. Use `dev/start[.bat]` instead. (#14285)

#### Contributors

- @jezdez
- @kenodegard

### 4.4.13 24.9.0 (2024-09-26)

#### Special announcement

This is an announcement about an important and positive **future** change in conda's functionality:

Following feedback from conda users about the pre-configuration of the conda code base to favor channels from Anaconda Inc, we've started the process to deprecate hardcoding Anaconda's channels as the default set of channels in the conda source code, which is a remnant of conda's incubation at the company.

In the future, we will rely on providers of conda distributions, such as miniforge or Anaconda (including miniconda), to pre-configure their preferred channels, e.g. by running the necessary `conda config --set channels` command.

We're also going to continue to work on improving channel management in the forseeable future and would love to get your feedback.

### Enhancements

- Add `conda.core.index.Index` as a faster drop-in replacement of the realized dictionary index. Note: The loggers are no longer implicitly initialized when fetching the index. Instead, you must explicitly call `conda.gateways.logging.initialize_logging`. (#13880, #14267)
- Alias `conda env remove` command to `conda remove --all`. (#13977)
- Add a new health check to `conda doctor` that detects if the `REQUESTS_CA_BUNDLE` env var points to a non-existent file. (#12905 via #14037)
- Add `--sha256` flag to `conda list --explicit` so it lists URLs with a SHA256 hash instead of MD5 and make `conda install|create` compatible with these inputs. (#2903, #7882 via #14048)
- Report `conda` version used to generate a `@EXPLICIT` text file. (#14048)
- Update health checks outputs for consistency. (#14049 via #14079)
- Fix Windows to Unix path conversion to handle UNC mounts and root paths. (#14157)
- Add a new `denylist_channels` config option and `CONDA_DENYLIST_CHANNELS` environment variable to explicitly deny using specific channels globally, which is complementing the already existing `allowlist_channels` config option. (#14176 via #14196)
- Add `conda commands` subcommand. (#14215)
- Add `conda.testing.fixtures.session_capsys`. Use this to capture stdout and stderr within module, package, and session scoped fixtures. (#14243)
- Add `conda.testing.fixtures.session_conda_cli`. Use this to invoke conda commands within module, package, and session scoped fixtures. (#14243)
- Add `conda.testing.fixtures.session_tmp_env`. Use this to create a conda environment within module, package, and session scoped fixtures. (#14243)

### Bug fixes

- Prevent directories that contain conda environments from being specified as an environment prefix when creating new envs; this provides guardrails against the accidental deletion of environments via commands such as `conda rename`. (#13955)
- Enable `conda doctor` to check whether the environment exists or not before trying to generate a health report. (#14042 via #14112)
- Fix `conda.common.configuration.ObjectParameter`'s ability to appropriately handle defaults. (#14148 via #14149)
- `conda remove [env] --all` command no longer deletes empty parent directories for environments that are removed. (#14173)
- Raise an error when attempting to remove non-existent environment. (#14174 via #14199)
- Add pre-/post-command hooks to activation subcommands (`activate`, `deactivate`, `reactivate`, `hooks`, and `commands`). (#14211 via #14212)
- Replace extensions from end of filename only, not `str.replace()`, in two places. (#14241)
- Fix PowerShell activation/deactivation to properly unset environment variables using `$Env:VARIABLE = $null` instead of `$Env:VARIABLE = ""`. (#14237 via #14246)

### Deprecations

- Mark `conda.core.index.get_index` as pending deprecation. Use `conda.core.index.Index` instead. (#13880)
- Mark `conda.core.index.get_reduced_index` as pending deprecation. Use `conda.core.index.ReducedIndex` instead. (#13880)
- Mark `conda.core.index.fetch_index` as pending deprecation. Use `conda.core.index.Index` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_prefix` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_cache` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_features` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._supplement_index_with_system` as pending deprecation. Use `conda.core.index.Index.reload` instead. (#13880)
- Mark `conda.core.index._make_virtual_package` as pending deprecation. Use `conda.models.records.PackageRecord.virtual_package` instead. (#13880)
- Mark `conda.core.subdir_data.make_feature_record` as pending deprecation. Use `conda.models.records.PackageRecord.feature` instead. (#13880)
- Mark `conda.plugins.manager.CondaPluginManager.get_virtual_packages` as pending deprecation. Use `conda.plugins.manager.CondaPluginManager.get_virtual_package_records` instead. (#13880)
- Mark `conda.misc.explicit(index_args)` as pending deprecation. (#14267)
- Mark `conda.cli.main_rename.validate_destination` as pending deprecation. Use `conda.cli.install.validate_new_prefix` instead. (#13955)
- Mark `conda.cli.main_env_remove.execute` as pending deprecation. Use `conda.cli.main_remove.execute` instead. (#13977)
- Mark `conda.activate.path_identity` as pending deprecation. Use `conda.common.path.path_identity` instead. (#14068)
- Mark `conda.utils.path_identity` as pending deprecation. Use `conda.common.path.path_identity` instead. (#14068)
- Mark `conda.models.leased_path_entry` as pending deprecation. (#14077)
- Postpone removal of `conda.base.context.Context.conda_exe` to `conda 25.3`. (#14077)
- Postpone removal of `conda.cli.python_api` to `conda 25.9`. (#14077)
- Remove `conda._vendor.appdirs`. Use `platformdirs` instead. (#14077)
- Remove `conda._vendor.cpuinfo`. (#14077)
- Remove `conda._vendor.distro`. Use `distro` instead. (#14077)
- Remove `conda.auxlib.collection.call_each`. (#14077)
- Remove `conda.auxlib.collection.firstitem`. (#14077)
- Remove `conda.auxlib.compat.NoneType`. (#14077)
- Remove `conda.auxlib.compat.primitive_types`. (#14077)

- Remove `conda.auxlib.compat.utf8_writer`. (#14077)
- Remove `conda.auxlib.exceptions.AssignmentError`. (#14077)
- Remove `conda.auxlib.exceptions.AuthenticationError`. (#14077)
- Remove `conda.auxlib.exceptions.InitializationError`. (#14077)
- Remove `conda.auxlib.exceptions.NotFoundError`. (#14077)
- Remove `conda.auxlib.exceptions.SenderError`. (#14077)
- Remove `conda.auxlib.type_coercion.boolify_truthy_string_ok`. (#14077)
- Remove `conda.auxlib.type_coercion.listify`. (#14077)
- Remove `conda.base.context.Context.root_dir`. Use `conda.base.context.Context.root_prefix` instead. (#14077)
- Remove `conda.base.exceptions`. (#14077)
- Remove `conda.cli.main.generate_parser`. Use `conda.cli.conda_argparse.generate_parser` instead. (#14077)
- Remove `conda.cli.main.init_loggers(context)` (#14077)
- Remove `conda.cli.main_rename.validate_src(name)`. (#14077)
- Remove `conda.cli.main_rename.validate_src(prefix)`. (#14077)
- Remove `conda.common.configuration.load_file_configs`. (#14077)
- Remove `conda.common.decorators.env_override`. (#14077)
- Remove `conda.common.decorators`. (#14077)
- Remove `conda.common.disk.temporary_content_in_file`. Use `tempfile` instead. (#14077)
- Remove `conda.core.package_cache_data.download`. Use `conda.gateways.connection.download.download` instead. (#14077)
- Remove `conda.core.package_cache_data.rm_fetched`. (#14077)
- Remove `conda.core.package_cache`. Use `conda.core.package_cache_data` instead. (#14077)
- Remove `conda.core.prefix_data.PrefixData._has_python`. (#14077)
- Remove `conda.core.subdir_data.get_cache_control_max_age`. Use `conda.gateways.repodata.get_cache_control_max_age` instead. (#14077)
- Remove `conda.exports.fetch_index`. Use `conda.core.index.fetch_index` instead. (#14077)
- Remove `conda.exports.IndexRecord`. Use `conda.models.records.PackageRecord` instead. (#14077)
- Remove `conda.gateways.anaconda_client.EnvAppDirs`. Use `platformdirs` instead. (#14077)
- Remove `conda.gateways.connection.adapters.ftp.FTPAdapter.stor`. (#14077)
- Remove `conda.gateways.connection.adapters.ftp.parse_multipart_files` (#14077)
- Remove `conda.gateways.logging.set_verbosity`. Use `conda.gateways.logging.set_log_level` instead. (#14077)
- Remove `conda.gateways.logging.VERBOSITY_LEVELS`. (#14077)
- Remove `conda.models.dist.IndexRecord`. Use `conda.models.records.PackageRecord` instead. (#14077)
- Remove `conda.models.enums.LeasedPathType`. (#14077)

- Remove `conda.models.leased_path_entry.LeasedPathEntry`. (#14077)
- Remove `conda.plugins.subcommands.doctor.get_prefix`. Use `conda.base.context.context.target_prefix` instead. (#14077)
- Remove `conda.plugins.subcommands.doctor.health_checks.display_health_checks`. (#14077)
- Remove `conda.plugins.subcommands.doctor.health_checks.display_report_heading`. (#14077)
- Remove `conda.testing.helpers.set_active_prefix`. Use `mocker.patch('conda.base.context.Context.active_prefix')` instead. (#14077)
- Remove `tests.env.utils.run_command`. Use `conda_cli` fixture instead. (#14077)
- Mark `tests.env.utils.make_temp_envs_dir` as pending deprecation. Use `tmp_envs_dir` fixture instead. (#14093)
- Mark `conda.gateways.disk.delete.rm_rf(max_retries)` as pending deprecated. (#14094)
- Mark `conda.gateways.disk.delete.rm_rf(trash)` as pending deprecated. (#14094)
- Mark `conda.gateways.disk.delete.try_rmdir_all_empty` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.gateways.disk.delete.move_to_trash` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.gateways.disk.delete.move_path_to_trash` as pending deprecated. Use `conda.gateways.disk.delete.rm_rf` instead. (#14094)
- Mark `conda.exports.move_to_trash` as pending deprecation. Use `conda.gateways.disk.delete.rm_rf` instead. (#14118)
- Mark `conda.activate.ensure_binary` as pending deprecation. Use `conda.common.compat.ensure_binary` instead. (#14144)
- Mark `conda.activate.ensure_fs_path_encoding` as pending deprecation. (#14144)
- Mark `conda.common.compat.six_with_metaclass` as pending deprecation. Use class' `metaclass=` keyword argument instead. (#14144)
- Mark `conda.common.compat.open` as pending deprecation. Use `conda.common.compat.open_utf8` instead. (#14144, #14169)
- Mark `conda.common.compat.ensure_unicode` as pending deprecation. (#14144)
- Mark `conda.common.compat.ensure_fs_path_encoding` as pending deprecation. (#14144)
- Mark `conda.common.compat.FILESYSTEM_ENCODING` as pending deprecation. (#14144, #14169)
- Mark `conda.activate.native_path_to_unix` as pending deprecation. Use `conda.common.path.win_path_to_unix` instead. (#14157)
- Mark `conda.activate.unix_path_to_native` as pending deprecation. Use `conda.common.path.unix_path_to_win` instead. (#14157)
- Mark `conda.activate._Cygpath` as pending deprecation. Use `conda.common.path._cygpath` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_UNIX` as pending deprecation. Use `conda.common.path._cygpath.RE_WIN_DRIVE` instead. (#14157)
- Mark `conda.activate._Cygpath.translate_unix` as pending deprecation. Use `conda.common.path._cygpath._to_unix_drive` instead. (#14157)

- Mark `conda.activate._Cygpath.RE_DRIVE` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_DRIVE` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_drive` as pending deprecation. Use `conda.common.path._cygpath._to_win_drive` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_MOUNT` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_MOUNT` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_mount` as pending deprecation. Use `conda.common.path._cygpath._to_win_mount` instead. (#14157)
- Mark `conda.activate._Cygpath.RE_ROOT` as pending deprecation. Use `conda.common.path._cygpath.RE_UNIX_ROOT` instead. (#14157)
- Mark `conda.activate._Cygpath.translation_root` as pending deprecation. Use `conda.common.path._cygpath._to_win_root` instead. (#14157)
- Mark `conda.utils.unix_path_to_win` as pending deprecation. Use `conda.common.path.unix_path_to_win` instead. (#14157)
- Deprecate `conda.env.specs.binstar` module. (#14158 via #14160)
- Deprecate `conda env [create|update] REMOTE_DEFINITION`. Use `conda env [create|update] --file=URL` instead. (#14158 via #14160)
- Mark `conda.testing.integration.BIN_DIRECTORY` as pending deprecation. Use `conda.common.path.BIN_DIRECTORY` instead. (#14188)
- Mark `conda.common.path.get_bin_directory_short_path()` as pending deprecation. Use `conda.common.path.BIN_DIRECTORY` instead. (#14188)
- Mark `conda.common.path.is_private_env_name` as pending deprecation. (#14189)
- Mark `conda.common.path.is_private_env_path` as pending deprecation. (#14189)
- Mark `conda shell.SHELL commands` as pending deprecation. Use `conda commands` instead. (#14215)
- The `defaults` multichannel will stop being the (implicit) default value for `channels`. Users relying on this behavior are encouraged to run `conda config --add channels defaults`. This is pending deprecation, and will be fully deprecated in 25.3. (#14178 via #14227)
- `conda config --add/--append channels ...` will warn when `defaults` is implicitly added. In `conda 25.3`, this behavior will be removed and users should run `conda config --add/--append channels defaults` explicitly if needed. Conda distribution installers like miniforge or miniconda will pre-configure conda channels during installation. (#12356 via #14227)
- Without an explicit `channels` configuration (via `condarc` files, environment variables, or CLI flags), conda will warn about using `defaults` implicitly. In 25.3, an empty list will be used. (#14227)
- Mark the `restore_free_channel` configuration option as pending deprecation. Add `https://repo.anaconda.com/pkgs/free` to your channel list after `defaults` instead. (#14231 via #14269)
- Mark `conda.testing.CondaCLIFixture` as pending deprecation. Use `conda.testing.fixtures.CondaCLIFixture` instead. (#14243)
- Mark `conda.testing.conda_cli` as pending deprecation. Use `conda.testing.fixtures.conda_cli` instead. (#14243)
- Mark `conda.testing.PathFactoryFixture` as pending deprecation. Use `conda.testing.fixtures.PathFactoryFixture` instead. (#14243)
- Mark `conda.testing.path_factory` as pending deprecation. Use `conda.testing.fixtures.path_factory` instead. (#14243)

- Mark `conda.testing.TmpEnvFixture` as pending deprecation. Use `conda.testing.fixtures.TmpEnvFixture` instead. (#14243)
- Mark `conda.testing.tmp_env` as pending deprecation. Use `conda.testing.fixtures.tmp_env` instead. (#14243)
- Mark `conda.testing.TmpChannelFixture` as pending deprecation. Use `conda.testing.fixtures.TmpChannelFixture` instead. (#14243)
- Mark `conda.testing.tmp_channel` as pending deprecation. Use `conda.testing.fixtures.tmp_channel` instead. (#14243)
- Mark `conda.testing.context_aware_monkeypatch` as pending deprecation. Use `conda.testing.fixtures.context_aware_monkeypatch` instead. (#14243)
- Mark `conda.testing.tmp_pkgs_dir` as pending deprecation. Use `conda.testing.fixtures.tmp_pkgs_dir` instead. (#14243)
- Mark `conda.testing.tmp_envs_dir` as pending deprecation. Use `conda.testing.fixtures.tmp_envs_dir` instead. (#14243)

### Docs

- Document `--platform` flag for `conda [env] create`. (#14100 via #14181)

### Other

- Update `xonsh` support to accomodate deprecated import path. (#14047)

### Contributors

- @anki-code made their first contribution in https://github.com/conda/conda/pull/14047
- @beeankha
- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @scw
- @SylvainCorlay made their first contribution in https://github.com/conda/conda/pull/14226
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

### 4.4.14 24.7.1 (2024-07-24)

#### Bug fixes

- Revert potential regression introduced in #13975. The previously forbidden characters (`^`, `%`, `!`, `=`, `(`, `)`, `\`) are allowed again until the impact is assessed. (#14065)

#### Contributors

- @jaimergp

### 4.4.15 24.7.0 (2024-07-17)

#### Enhancements

- Add a new `reporters` setting for configuring output. (#13736)
- Report traceback of plugin loading errors with verbosity 2 or higher (`-vv` or more). (#13742 via #13846)
- Skip checking for `.pyc` and `.pyo` files in the `conda doctor` "missing files" health check. (#13370 via #13931)
- **Breaking change** `conda list --explicit` will not print authentication details by default. A new flag `--auth` has been added so folks can opt-in to this behaviour. (#13936)
- Print transaction report for `@EXPLICIT` lockfile installs too. (#13940)
- Do not require `-n/--name` or `-p/--prefix` if `conda create` is invoked with `--dry-run`. (#13941)
- Add an `envvars_force_uppercase` setting which defaults to `True`, uppercasing all environment variables (thereby justifying `conda`'s current behaviour); when `envvars_force_uppercase` is set to `False`, conda will only save preserved-case variable names. (#13713 via #13943)
- Alias `conda env list` command to `conda info --envs`. (#13972)

#### Bug fixes

- Improve treatment of logger levels. (#13735)
- Mask authentication details in `conda-meta/*.json` metadata. (#13937)
- Mask Anaconda.org tokens in verbose logs. (#13939, #13987)
- Fix parsing error when history file only contains a single commented line. (#13960)
- Add missing `emscripten` and `wasi` entries to the recognized platforms, and `wasm32` to the recognized architectures. (#13095)
- Fix checksum comparisons in `conda.gateways.connection.download.download()` to be case insensitive. (#13969)
- Disallow some more characters in Windows for prefix names (`^`, `%`, `!`, `=`, `(`, `)`, `\`). These characters complicate or prevent environment activation if present. (#12558 via #13975)
- Fix caching when `repodata.json` contains `\r\n` line endings. (#14002 via #14003)
- Fix `conda.core.portability.binary_replace` not matching chunks that end with `\n`. (#14043 via #14044)

### Deprecations

- Mark `conda.gateways.logging.initialize_root_logger` as pending deprecation. (#13735, #14046)
- Mark `conda.cli.main_env_list.execute` as pending deprecation. Use `conda.cli.main_info.execute` instead. (#13972)
- Revert `--all` deprecation in `conda info`. (#14004)
- Mark `conda.exports.iteritems` as pending deprecation. Use builtin `dict.items()` instead. (#14034)
- Mark `conda.exports.Completer` as pending deprecation. (#14034)
- Mark `conda.exports.InstalledPackages` as pending deprecation. (#14034)
- Mark `conda.exports.KEYS` as pending deprecation. (#14034)
- Mark `conda.exports.KEYS_DIR` as pending deprecation. (#14034)
- Mark `conda.exports.hash_file` as pending deprecation. (#14034)
- Mark `conda.exports.verify` as pending deprecation. (#14034)
- Mark `conda.exports.symlink_conda` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.exports._symlink_conda_hlp` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.exports.win_conda_bat_redirect` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.win_path_to_cygwin` as pending deprecation. Use `conda.common.path.win_path_to_unix` instead. (#14034)
- Mark `conda.utils.cygwin_path_to_win` as pending deprecation. Use `conda.utils.unix_path_to_win` instead. (#14034)
- Mark `conda.utils.translate_stream` as pending deprecation. (#14034)
- Mark `conda.utils.unix_shell_base` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.msys2_shell_base` as pending deprecation. Use `conda.activate` instead. (#14034)
- Mark `conda.utils.shells` as pending deprecation. Use `conda.activate` instead. (#14034)

### Docs

- Clarify proxy server configuration in documentation. (#12856)
- Add type hints and doc strings to `conda.core.envs_manager`. (#13817)
- Add logging overview as deep-dive. (#13735)
- Update conda cheatsheet text and add it directly to cheatsheet page. (#13889)
- Add cheatsheet PDF download to cheatsheet page. (#13909)
- Add `ssl_verify: truststore` to the user guide. (#13935)
- Fix the help text of the `satisfied-skip-solve` flag. (#13946)
- Add a section explaining how to correctly raise exceptions from a plugin. (#13741 via #13950)
- Add new article for configuring `envs_dirs` and `pkgs_dirs`. (#13954)

### Other

- Replace calls to `logger.warn` with `logger.warning`. (#13963)

### Contributors

- @beeankha
- @conda-bot
- @erik-whiting made their first contribution in https://github.com/conda/conda/pull/13877
- @ifitchet
- @jaimergp
- @jezdez
- @kathatherine
- @kelvinou01 made their first contribution in https://github.com/conda/conda/pull/14044
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @Nathann03
- @zeehio
- @skupr-anaconda made their first contribution in https://github.com/conda/conda/pull/13946
- @tl-hbk
- @DerThorsten made their first contribution in https://github.com/conda/conda/pull/13962
- @travishathaway
- @dependabot[bot]
- @padeoe made their first contribution in https://github.com/conda/conda/pull/12856
- @pre-commit-ci[bot]

## 4.4.16 24.5.0 (2024-05-08)

### Enhancements

- Report which `MatchSpec` item caused `Invalid*Spec` exceptions for more informative error messages. (#11203 via #13598)
- MSYS2 packages can now use the upstream installation prefixes. (#13649)
- Add support for CEP-15 `base_url` field in `repodata.json`. (#13137 via #13744)
- In custom channel settings, allow specification of channel URLs using a glob-like wildcard pattern, e.g. for user with auth handler plugins. (#13778 via #13779)

### Bug fixes

- Fix `conda notices --json` to correctly output JSON. (#13561)
- Fix prefix replacement for Windows `subdir` on Unix. (#13689)

### Deprecations

- Mark `conda.plan._get_best_prec_match` as pending deprecation. Use `conda.misc._get_best_prec_match` instead. (#12421)
- Mark `conda.plan._handle_menuinst` as pending deprecation. (#12421)
- Mark `conda.plan._inject_UNLINKLINKTRANSACTION` as pending deprecation. (#12421)
- Mark `conda.plan._plan_from_actions` as pending deprecation. (#12421)
- Mark `conda.plan.add_defaults_to_specs` as pending deprecation. (#12421)
- Mark `conda.plan.add_unlink` as pending deprecation. (#12421)
- Mark `conda.plan.display_actions` as pending deprecation. (#12421)
- Mark `conda.plan.execute_actions` as pending deprecation. (#12421)
- Mark `conda.plan.get_blank_actions` as pending deprecation. (#12421)
- Mark `conda.plan.install_actions` as pending deprecation. (#12421)
- Mark `conda.plan.print_dists` as pending deprecation. (#12421)
- Mark `conda.plan.revert_actions` as pending deprecation. Use `conda.cli.install.revert_actions` instead. (#12421)
- Mark `conda.plan` as an entrypoint as pending deprecation. (#12421)
- Mark `conda.activate._Activator.add_export_unset_vars` as pending deprecation. Use `conda.activate._Activator.get_export_unset_vars` instead. (#13720)
- Mark `conda.activate._Activator.get_scripts_export_unset_vars` as pending deprecation. Use `get_scripts_export_unset_vars` helper function in `test_activate.py` instead. (#13720)
- Mark `conda.activate._Activator._get_path_dirs(extra_library_bin)` as pending deprecation. (#13720)
- Mark `conda.activate.JSONFormatMixin.get_scripts_export_unset_vars` as pending deprecation. Use `conda.activate._Activator.get_export_unset_vars` instead. (#13720)
- Mark `conda.gateways.logging.trace` as pending deprecation. Use `Logger.log(conda.common.constants.TRACE, msg)` instead. (#13732)
- Mark `conda create --mkdir` as pending deprecation. The argument is redundant and unnecessary. (#13751)
- Mark `conda install --mkdir` as pending deprecation. Use `conda create` instead. (#13751)
- Mark `conda._vendor.frozendict` as pending deprecation. Use `frozendict` instead. (#13767 via #13766)
- Mark `conda.auxlib.collection.make_immutable` as pending deprecation. Use `frozendict.deepfreeze` instead. (#13801)
- Mark `conda.plan.execute_plan` as pending deprecation. (#13869)
- Mark `conda.plan.execute_instructions` as pending deprecation. (#13869)
- Mark `conda.plan._update_old_plan` as pending deprecation. (#13869)

### Docs

- Add type hints and doc strings to `conda.core.index`. (#13816)

### Other

- Remove `setuptools` remainings (`MANIFEST.in`, `wheel` build dependency) not required since the move to `hatch` in #12509. (#13684)
- Remove and update any imports inside conda that is importing from `conda/exports.py`. (#13869)

### Contributors

- @beeankha
- @conda-bot
- @dbast
- @ifitchet made their first contribution in https://github.com/conda/conda/pull/13649
- @isuruf
- @jaimergp
- @jezdez
- @kenodegard
- @zklaus
- @ForgottenProgramme
- @mattkram
- @Nathann03 made their first contribution in https://github.com/conda/conda/pull/13816
- @dwr-psandhu made their first contribution in https://github.com/conda/conda/pull/13770
- @travishathaway
- @dependabot[bot]
- @pre-commit-ci[bot]

## 4.4.17 24.4.0 (2024-04-24)

### Enhancements

- For Windows users, the stub executables used for Python entrypoints in packages are now codesigned. (#13721)

### Contributors

- @dholth
- @jezdez
- @Callek made their first contribution in https://github.com/conda/conda/pull/13721

## 4.4.18 24.3.0 (2024-03-12)

### Enhancements

- Show first few characters of undecodeable response if `repodata.json` raises `JSONDecodeError`. (#11804)
- Update `conda.gateways.subprocess.subprocess_call` to use `text=True` to avoid manual encoding/decoding. (#13240)
- Add a new plugin hook giving plugin authors the ability to define new settings. (#13554)
- Optimize module imports to speed up `conda activate`. (#13567 via #13568)
- Move `conda env export` to `conda export` and alias the old command to the new command. (#13577)
- Report progress while running `conda install --revision <idx>`. (#13611)
- Add `conda.testing.tmp_channel` pytest fixture to create a temporary local channel for testing. (#13634)

### Bug fixes

- Print traceback on `KeyboardInterrupt` instead of raising another `AttributeError` exception, when conda debugging logs are enabled. (#13531)
- Parse integer channel notice IDs as `str` instead of raising an exception. (#13543)
- Add direct runtime dependency on `zstandard` for use when downloading `repodata.json.zst`. (#13551)
- Fallback to `repodata.json` if `repodata.json.zst` cannot be decompressed as `zstandard`. (#13558)
- `conda rename` command no longer throws an error when conda is not active. (#13565)
- Fallback to `repodata.json` from `repodata.json.zst` on most 4xx error codes. (#13573)
- Fix excess resource usage by log handling when fetching repodata. (#13541 via #13628)
- Re-enable `--subdir` and `--platform` flags to be available for `conda env create` command. (#13632)
- Fix `__archspec` virtual package on Windows to return microarchitecture instead of the default `x86_64`. (#13641)
- Check `Content-Length` is nonzero before calculating progress, avoiding a possible `ZeroDivisionError`. (#13653, #13671)

### Deprecations

- Discontinue custom docker images. Use images provided by Anaconda Inc. or conda-forge instead. (#13162)
- Mark `conda.common.compat.encode_arguments` as pending deprecation. (#13240)
- Remove `conda.export.handle_proxy_407`. (#13629)
- Mark `conda.testing.integration.make_temp_channel` as pending deprecation. Use `conda.testing.tmp_channel` fixture instead. (#13634)
- Mark `conda.testing.integration.running_a_python_capable_of_unicode_subprocessing` as pending deprecation. (#13634)
- Mark `conda.testing.integration.set_tmpdir` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration._get_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration.make_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration.FORCE_temp_prefix` as pending deprecation. Use `tmp_path`, `conda.testing.path_factory`, or `conda.testing.tmp_env` instead. (#13634)
- Mark `conda.testing.integration.create_temp_location` as pending deprecation. Use `tmp_path` or `conda.testing.path_factory` instead. (#13634)
- Mark `conda.testing.integration.tempdir` as pending deprecation. Use `tmp_path` or `conda.testing.path_factory` instead. (#13634)
- Mark `conda.testing.integration.reload_config` as pending deprecation. Use `conda.base.context.reset_context` instead. (#13634)
- Postpone `conda.base.context.Context.conda_exe` deprecation to `conda 24.9`. (#13634)
- Postpone `conda.testing.integration.run_command` deprecation to `conda 25.3`. (#13634)
- Postpone loading subcommands from executables deprecation to `conda 25.3`. (#13634)
- Remove vendored `conda._vendor.boltons`. Use `boltons` package instead. (#12681 via #13634)
- Remove `conda.auxlib.packaging`. Use a modern build system instead; see https://packaging.python.org/en/latest/tutorials/packaging-projects#creating-pyproject-toml for more details. (#12681 via #13634)
- Remove `conda env create --force`. Use `conda env create --yes` instead. (#12681 via #13634)
- Remove `conda info PACKAGE`. Use `conda search PACKAGE --info` instead. (#12681 via #13634)
- Remove `conda.core.subdir_data.fetch_repodata_remote_request`. Use `conda.core.subdir_data.SubdirData.repo_fetch.fetch_latest_parsed` instead." (#12681 via #13634)
- Remove `conda.exports.memoized`. Use `functools.lru_cache` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read._digest_path`. Use `conda.gateways.disk.read.compute_sum` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read.compute_md5sum`. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12681 via #13634)
- Remove `conda.gateways.disk.read.compute_sha256sum`. Use `conda.gateways.disk.read.compute_sum(path, "sha256")` instead. (#12681 via #13634)
- Remove `conda.instructions.PREFIX`. (#12681 via #13634)

- Remove `conda.instructions.PREFIX_CMD`. (#12681 via #13634)
- Remove `conda.testing.encode_for_env_var`. (#12681 via #13634)
- Remove `conda.testing.conda_check_versions_aligned`. (#12681 via #13634)
- Remove `conda.testing.helpers.run_inprocess_conda_command`. Use `conda.testing.tmp_env` instead. (#12681 via #13634)
- Remove `conda.testing.helpers.capture_json_with_argv`. (#12681 via #13634)
- Remove `conda.testing.integration.get_conda_list_tuple`. Use `conda.core.prefix_data.PrefixData.get` instead. (#12681 via #13634)
- Remove `conda.utils.md5_file`. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12681 via #13634)
- Remove `conda.utils.hashsum_file`. Use `conda.gateways.disk.read.compute_sum` instead. (#12681 via #13634)
- Remove `conda.utils.safe_open`. Use `open` instead. (#12681 via #13634)
- Remove `python -m conda_env`. Use `conda env` or `python -m conda env` instead. (#12681 via #13634)
- Remove `conda_env.env.load_from_directory`. (#12681 via #13634)
- Remove `conda_env.pip_util.get_pip_version`. (#12681 via #13634)
- Remove `conda_env.pip_util.PipPackage`. (#12681 via #13634)
- Remove `conda_env.pip_util.installed`. (#12681 via #13634)
- Remove `conda_env.pip_util._canonicalize_name`. (#12681 via #13634)
- Remove `conda_env.pip_util.add_pip_installed`. (#12681 via #13634)

### Docs

- Update the navigation links for Miniconda. (#13572)

### Other

- Remove `dev/*` scripts in favor of `conda-incubator/setup-miniconda` GitHub Action in `.github/workflows/tests.yml`. (#13162)
- Stop chaining commands for steps in `.github/workflows/tests.yml`. (#12418 via #13162)
- Modernize tests. (#13547, #13292)
- Run GitHub tests workflow also on `osx-arm64` (aka Apple Silicon) runners. Enable `osx-arm64` canary builds. Fix or disable broken tests. (#13617)
- Upload stable release artifacts to GitHub releases during releases. (#13399)

### Contributors

- @beeankha
- @conda-bot
- @dbast
- @dholth
- @FFY00
- @isuruf
- @jaimergp
- @jezdez
- @jjhelmus
- @kenodegard
- @zklaus made their first contribution in https://github.com/conda/conda/pull/13579
- @ForgottenProgramme
- @mbargull
- @travishathaway
- @pre-commit-ci[bot]

## 4.4.19 24.1.2 (2024-02-15)

### Bug fixes

- Fix deprecated `fetch_repodata_remote_request` when `repodata_use_zst` is enabled. (#13595)

### Contributors

- @dholth

## 4.4.20 24.1.1 (2024-02-12)

### Bug fixes

- Fallback to `repodata.json` if `repodata.json.zst` cannot be decompressed as zstandard. (#13558)
- Fallback to `repodata.json` from `repodata.json.zst` on most 4xx error codes. (#13573)

### Contributors

- @dholth

## 4.4.21 24.1.0 (2024-01-24)

### Special announcement

#### The `conda_env.*` modules have been merged into the `conda` package!

To improve the integration of the `conda env` subcommand (previously standalone), we've moved its code into the `conda` package, while allowing old `conda env` commands to still work via Python import redirects. This is a first step of many to improving the user experience of the conda command line interface related to environment management. (#13168)

### Enhancements

- Verify signatures on to-be-installed packages instead of on all packages. (#11545, #13053)
- Add new `pre-solves` and `post-solves` plugin hooks. (#13053)
- Add support for Python 3.12. (#13072)
- Check `repodata.json.zst` for faster repodata downloads. (#13256)
- Add `--skip-flexible-search` option in `conda search` to skip flexible search. (#13315)
- Provide a more useful warning when attempting to rename a non-existent prefix. (#13387)
- Add a new flag `--keep-env` to be used with `conda remove --all`. It allows users to delete all packages in the environment while retaining the environment itself. (#13419)
- Add a Y/N prompt warning users that `conda env remove` and `conda remove --all` deletes not only the conda packages but the entirety of the specified environment. (#13440)
- Add `--repodata-use-zst/--no-repodata-use-zst` flag to control `repodata.json.zst` check; corresponding `repodata_use_zst: true/false` for `.condarc`. Default is to check for `repodata.json.zst`. Disable if remote returns unparseable `repodata.json.zst` instead of correct data or 404. (#13504)

### Bug fixes

- Create the `~/.conda` directory before trying to write to the `environments.txt` file. (#13338)
- Ensure `PackageRecord.timestamp` is dumped in milliseconds. (#13483)
- Fix error when setting a non-default repodata filename via `CONDA_REPODATA_FNS`. (#13490)
- Fix the config file location where the integrated Anaconda client gateway loads user configuration from. This is a regression introduced in conda 23.11.0 when the `platformdirs` library was adopted. (#13517 via #13520)
- Interpret missing `Cache-Control` header as `max-age=0` instead of exception. (#13522)

### Deprecations

- Mark `conda_env/cli/common` as pending deprecation. Use `conda.cli.common` instead. (#13168)
- Mark `conda_env/cli/main_config` as pending deprecation. Use `conda.cli.main_env_config` instead. (#13168)
- Mark `conda_env/cli/main_create` as pending deprecation. Use `conda.cli.main_env_create` instead. (#13168)
- Mark `conda_env/cli/main_export` as pending deprecation. Use `conda.cli.main_env_export` instead. (#13168)
- Mark `conda_env/cli/main_list` as pending deprecation. Use `conda.cli.main_env_list` instead. (#13168)
- Mark `conda_env/cli/main_remove` as pending deprecation. Use `conda.cli.main_env_remove` instead. (#13168)
- Mark `conda_env/cli/main_update` as pending deprecation. Use `conda.cli.main_env_update` instead. (#13168)
- Mark `conda_env/cli/main_vars` as pending deprecation. Use `conda.cli.main_env_vars` instead. (#13168)
- Mark `conda_env/env` as pending deprecation. Use `conda.env.env` instead. (#13168)
- Mark `conda_env/installers/base` as pending deprecation. Use `conda.env.installers.base` instead. (#13168)
- Mark `conda_env/installers/conda` as pending deprecation. Use `conda.env.installers.conda` instead. (#13168)
- Mark `conda_env/installers/pip` as pending deprecation. Use `conda.env.installers.pip` instead. (#13168)
- Mark `conda_env/pip_util` as pending deprecation. Use `conda.env.pip_util` instead. (#13168)
- Mark `conda_env/specs` as pending deprecation. Use `conda.env.specs` instead. (#13168)
- Mark `conda_env/specs/binstar` as pending deprecation. Use `conda.env.specs.binstar` instead. (#13168)
- Mark `conda_env/specs/requirements` as pending deprecation. Use `conda.env.specs.requirements` instead. (#13168)
- Mark `conda_env/specs/yaml_file` as pending deprecation. Use `conda.env.specs.yaml_file` instead. (#13168)
- Mark `conda.testing.integration.make_temp_package_cache` as pending deprecation. (#13511)

### Docs

- Update Getting Started documentation in User Guide. (#13190)
- Add GoatCounter (https://www.goatcounter.com/) as an analytics tool. (#13384)
- Add type hints and doc strings to `conda.cli.main_info`. (#13445)
- Add type hints and doc strings to `conda.cli.main_search`. (#13465)

### Other

- Add type hinting for `VersionOrder` class. (#13380)
- Re-enable and apply `pyupgrade` via `ruff`. (#13272, #13433)
- Start tracking performance in continuous integration and automatically report about it in pull requests. (#13460)
- Add `tmp_pkgs_dir` fixture to replace `make_temp_package_cache`. (#13511)
- Improve lock API for the repodata cache. (#13455)

### Contributors

- @beeankha
- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @jezdez
- @johnnynunez
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @marcoesters
- @mfansler
- @schuylermartin45 made their first contribution in https://github.com/conda/conda/pull/13385
- @travishathaway
- @pre-commit-ci[bot]
- @samhaese made their first contribution in https://github.com/conda/conda/pull/13465

## 4.4.22 23.11.0 (2023-11-30)

### Special announcement

New `menuinst` v2 support!

`conda` has supported Start menu items on Windows for a long time. This is what allows users to open up their Miniconda prompt on CMD (Command Prompt) with an initialized `conda` session. This menu item (or shortcut) creation logic is provided by the `menuinst` package.

With the release of 23.11.0, `conda` now supports `menuinst` v2, which enables the same experience across Windows, Linux, and macOS. This means package builders will be able to provide desktop icons across all operating systems, which can be especially useful for GUI applications. See the documentation for more details.

If you don't want `conda` to create shortcuts, you can disable it via:

- `shortcuts: false` entry in your `.condarc` configuration

- `CONDA_SHORTCUTS=false` environment variable
- `--no-shortcuts` command-line flag

### Enhancements

- Add support for `menuinst` v2, enabling shortcuts across all platforms (Windows, Linux, macOS) using a new JSON schema (see CEP-11). Retain support for old v1-style JSON menus. (#11882)
- Stop using vendored `chardet` package by `requests`/`pip`; explicitly depend on `charset_normalizer`. (#13171)
- Introduce a new plugin hook, `CondaHealthCheck`, as part of `conda doctor`. (#13186)
- Include `activate` and `deactivate` in the `--help` command list. (#13191)
- Prioritize download of larger packages to prevent smaller ones from waiting. (#13248)
- Display the used solver in `conda info` output for debugging purposes. (#13265)
- Add `__conda` virtual package. (#13266)
- Switch from `appdirs` to `platformdirs`. (#13306)
- Implement resume capability for interrupted package downloads. (#8695)

### Bug fixes

- Log expected JLAP range-request errors at `info` level, occurring when the remote file has rolled over. (#12913)
- Fix a bug causing an error when options like `--debug` are used without specifying a command. (#13232)
- Improve CTRL-C (cancellation) handling for parallel download threads. (#13234)
- Allow overriding of `CONDA_FETCH_THREADS`/`fetch_threads` to set parallel package downloads beyond the default `5`. (#13263)
- Require `requests >=2.28` for enhanced `response.json()` exception handling. (#13346)
- Apply `callback=reset_context` in `conda.plan` to resolve `conda-build` + `conda-libmamba-solver` incompatibilities. (conda-libmamba-solver#393 and conda-libmamba-solver#386 via #13357)

### Deprecations

- Deprecate `conda.plugins.subcommands.doctor.health_checks.display_health_checks` function. (#13186)
- Deprecate `conda.plugins.subcommands.doctor.health_checks.display_report_heading` function. (#13186)
- Remove `ruamel_yaml` fallback; use `ruamel.yaml` exclusively. (#13218)
- Deprecate `conda.gateways.anaconda_client.EnvAppDirs` in favor of `platformdirs`. (#13306)
- Mark `conda._vendor.cpuinfo` for pending deprecation. (#13313)
- Deprecate `conda._vendor.distro` in favor of the `distro` package. (#13317)

### Docs

- Add the `conda-sphinx-theme` to the conda documentation. (#13298)
- Update specific pages to remove redundant TOC entries. (#13298)
- Include instructions on updating `conda` in the main `README.md`. (#13343)

### Other

- Add a lighter weight s3 test; update embedded test package index. (#13085)
- Refactor code to use lazy imports for all relative imports in `conda.cli.main_*`, and separate argparse configuration functions from `conda.cli.conda_argparse` to their respective `conda.cli.main_*` modules. (#13173)
- Move custom `argparse.Actions` to `conda.cli.actions` (e.g., `NullCountAction`), and relocate helper argparse functions to `conda.cli.helpers` (e.g., `add_parser_prefix`). (#13173)
- Update upper bound for `ruamel.yaml` to `<0.19` following the release of `0.18`. (#13258)
- Replace `black` with `ruff format` in pre-commit. (#13272)

### Contributors

- @AniketP04 made their first contribution in https://github.com/conda/conda/pull/13224
- @beeankha
- @13rac1 made their first contribution in https://github.com/conda/conda/pull/13191
- @conda-bot
- @dholth
- @eltociear
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @kennethlaskoski made their first contribution in https://github.com/conda/conda/pull/13322
- @ForgottenProgramme
- @marcoesters
- @opoplawski
- @scruel made their first contribution in https://github.com/conda/conda/pull/13274
- @travishathaway
- @gfggithubleet made their first contribution in https://github.com/conda/conda/pull/13270
- @pre-commit-ci[bot]

### 4.4.23 23.10.0 (2023-10-30)

#### Special announcement

This is an announcement about an important change in conda's functionality:

#### With this 23.10.0 release we are changing the default solver of conda to `conda-libmamba-solver`!

The previously "classic" solver is based on pycosat/Picosat and will remain part of conda for the foreseeable future, a fallback is possible and available.

#### Why are we switching the solver?

In short: to make conda faster and more accurate.

A "solver" is the core component of most package managers; it calculates which dependencies (and which version of those dependencies) to install when a user requests to install a package from a package repository. To address growth-related challenges within the conda ecosystem, the conda maintainers, alongside partners Anaconda, Quansight and QuantStack, introduced a new conda dependency solver based on the Mamba project in December 2022.

Since July 2023, the `conda-libmamba-solver` plugin has been included in all major conda ecosystem installers (miniforge, miniconda, mambaforge and Anaconda Distribution), but was disabled by default. As soon as these installers are updated to contain conda 23.10.0 or later, they will automatically default to using the conda-libmamba-solver plugin.

#### What can I do if this update doesn't work for me?

If the new solver is not working as you expect:

- Check if the behavior you are observing is a known issue or a deliberate change.
- If that's not the case, please consider submitting a bug report or feature request in the conda-libmamba-solver repository.
- If necessary, you can go back to using the `classic` solver without modifying your conda installation:
  - When possible, pass the command line option `--solver=classic` to your `conda` calls.
  - Otherwise (e.g. for `conda build ...` or `constructor ...`), set the environment variable `CONDA_SOLVER=classic`.
  - For permanent changes, use the conda configuration system: `conda config --set solver classic`.

#### Where can I learn more about conda-libmamba-solver?

The documentation of the `conda-libmamba-solver` plugin can be found on conda.github.io/conda-libmamba-solver.

For more information about the `conda-libmamba-solver` rollout plan, please also see our blog post from earlier this year.

### Enhancements

- Provide `--platform` and `--subdir` flags to create environments for non-native platforms, remembering that choice in future operations. (#11505 via #11794)
- IMPORTANT: Set `solver: libmamba` as the new default solver. (#12984)

### Bug fixes

- Check name of symlink, not its target against valid configuration file names (`condarc`, `.condarc`, or `*.yml/yaml`). (#12956)
- Have `conda doctor` ignore blank lines in `~/.conda/environments.txt`. (#12984)

### Deprecations

- Mark `conda.cli.main.generate_parser` as pending deprecation. Use `conda.cli.conda_argparse.generate_parser` instead. (#13144)
- Mark `conda.auxlib.collection.firstitem` as pending deprecation. (#13144)
- Mark `conda.auxlib.collection.call_each` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.NoneType` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.primative_types` as pending deprecation. (#13144)
- Mark `conda.auxlib.compat.utf8_writer` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.AuthenticationError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.NotFoundError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.InitializationError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.SenderError` as pending deprecation. (#13144)
- Mark `conda.auxlib.exceptions.AssignmentError` as pending deprecation. (#13144)
- Mark `conda.auxlib.type_coercion.boolify_truthy_string_ok` as pending deprecation. (#13144)
- Mark `conda.auxlib.type_coercion.listify` as pending deprecation. (#13144)
- Mark `conda.models.dist.IndexRecord` as pending deprecation for removal in 24.9. (#13193)
- Mark `conda.exports.fetch_index` as pending deprecation for removal in 24.9. Use `conda.core.index.fetch_index` instead. (#13194)

### Other

- Constrain minimum conda-build version to `>=3.27`. (#13177)

#### Contributors

- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @kenodegard
- @timhoffm
- @pre-commit-ci[bot]

### 4.4.24 23.9.0 (2023-09-27)

#### Special announcement

This is an announcement about an important and positive future change in conda's functionality:

> We will change the default solver of conda to `conda-libmamba-solver` in a **special 23.10.0 release** in the near future!

You can already benefit from it *today* by configuring your conda installation to use it (e.g. by running `conda config --set solver libmamba`).

The current "classic" solver is based on pycosat/Picosat and will remain part of conda for the foreseeable future, a fallback is possible and available (see below).

#### Plan to change the default solver

Here is our updated plan to change the default solver, to better follow CEP 8 and reduce the potential impact on conda users:

- The upcoming, special 23.10.0 release will be dedicated to the switch of the default solver to `libmamba`.
- Users will be able to opt out of the `libmamba` solver and use the `classic` solver instead, by using one of these options:
  - the `--solver=classic` command line option,
  - the `CONDA_SOLVER=classic` environment variable or
  - running `conda config --set solver classic`.
- All development of `conda-libmamba-solver` plugin happens in the conda-libmamba-solver repo, including issue tracking.
- The documentation of the `conda-libmamba-solver` plugin can be found on conda.github.io/conda-libmamba-solver.

For more information about the `conda-libmamba-solver` rollout plan, please also see our blog post from earlier this year.

### Context

A "solver" is the core component of most package managers; it calculates which dependencies (and which version of those dependencies) to install when a user requests to install a package from a package repository. To address growth-related challenges within the conda ecosystem, the conda maintainers, alongside partners Anaconda, Quansight and QuantStack, introduced a new conda dependency solver based on the Mamba project in December 2022.

Since July 2023, that `conda-libmamba-solver` plugin has been included in and automatically installed with all major conda ecosystem installers (miniforge, miniconda, mambaforge and Anaconda Distribution), *with the default solver configuration unchanged*.

### Enhancements

- Improve speed of `fish` shell initialization. (#12811)
- Directly suppress use of binstar (conda) token when fetching trust metadata. (#12889)
- Add a new "auth handler" plugin hook for conda. (#12911)
- Lock index cache metadata by default. Added `--no-lock` option in case of problems, should not be necessary. Older `--experimental=lock` no longer has an effect. (#12920)
- Add `context.register_envs` option to control whether to register environments in `~/.conda/environments.txt` when they are created. Defaults to true. (#12924)
- Inject a new detailed output verbosity level (i.e., the old debug level `-vv` is now `-vvv`). (#12985, #12977, #12420, #13036)
- Add support for `truststore` to the `ssl_verify` config option, enabling conda to use the operating system certificate store (requires Python 3.10 or later). (#13075 and #13149)
- Add `emscripten-wasm32` and `wasi-wasm32` platforms to known platforms. (#13095)
- Adds the `py.typed` marker file to the `conda` package for compliance with PEP-561. (#13107)
- Import `boto3` only when S3 channels are used, saving startup time. (#12914)

### Bug fixes

- When using pip dependencies with `conda env create`, check the directory permissions before writing to disk. (#11610)
- Hide `InsecureRequestWarning` for JLAP when `CONDA_SSL_VERIFY=false`, matching non-JLAP behavior. (#12731)
- Disallow ability to create a conda environment with a colon in the prefix. (#13044)
- Fix `AttributeError` logging response with nonexistent request when using JLAP with `file:///` URIs. (#12966)
- Do not show progress bars in non-interactive runs for cleaner logs. (#12982)
- Fix S3 bucket name. (#12989)
- Default `--json` and `--debug` to `NULL` so as to not override `CONDA_JSON` and `CONDA_DEBUG` environment variables. (#12987)
- `XonshActivator` now uses `source-bash` in non-interactive mode to avoid side-effects from interactively loaded RC files. (#13012)
- Fix `conda remove --all --json` output. (#13019)

- Update test data to stop triggering security scanners' false-positives. (#13034)
- Fix performance regression of basic commands (e.g., `conda info`) on WSL. (#13035)
- Configure conda to ignore "Retry-After" header to avoid the scenarios when this value is very large and causes conda to hang indefinitely. (#13050)
- Treat `JSONDecodeError` on `repodata.info.json` as a warning, equivalent to a missing `repodata.info.json`. (#13056)
- Fix sorting error for `conda config --show-sources --json`. (#13076)
- Catch `OSError` in `find_commands` to account for incorrect `PATH` entries on Windows. (#13125)
- Catch a `NotWritableError` when trying to find the first writable package cache dir. (#9609)
- `conda env update --prune` uses only the specs coming from `environment.yml` file and ignores the history specs. (#9614)

### Deprecations

- Removed `conda.another_unicode()`. (#12948)
- Removed `conda._vendor.toolz`. (#12948, #13141)
- Removed `conda._vendor.tqdm`. (#12948)
- Removed `conda.auxlib.decorators.memoized` decorator. (#12948)
- Removed `conda.base.context.Context.experimental_solver`. (#12948)
- Removed `conda.base.context.Context.conda_private`. (#12948)
- Removed `conda.base.context.Context.cuda_version`. (#12948)
- Removed `conda.base.context.get_prefix()`. (#12948)
- Removed `conda.cli.common.ensure_name_or_prefix()`. (#12948)
- Removed `--experimental-solver` command line option. (#12948)
- Removed `conda.common.cuda` module. (#12948)
- Removed `conda.common.path.explode_directories(already_split)`. (#12948)
- Removed `conda.common.url.escape_channel_url()`. (#12948)
- Removed `conda.core.index.check_whitelist()`. (#12948)
- Removed `conda.core.solve._get_solver_class()`. (#12948)
- Removed `conda.core.subdir_data.read_mod_and_etag()`. (#12948)
- Removed `conda.gateways.repodata.RepodataState.load()`. (#12948)
- Removed `conda.gateways.repodata.RepodataState.save()`. (#12948)
- Removed `conda.lock` module. (#12948)
- Removed `conda_env.cli.common.stdout_json()`. (#12948)
- Removed `conda_env.cli.common.get_prefix()`. (#12948)
- Removed `conda_env.cli.common.find_prefix_name()`. (#12948)
- Remove import of deprecated cgi module by deprecating ftp STOR support. (#13013)

- Require `boto3` for S3 support and drop support for the older `boto` as it doesn't support our minimum required version of Python. (#13112)
- Reduce startup delay from deprecations module by using `sys._getframe()` instead of `inspect.stack()`. (#12919)

### Other

- Use Ruff linter in pre-commit configuration (#12279)
- Remove unused `cache_path` arguments from `RepoInterface`/`JlapRepoInterface`; replaced by cache object. (#12927)

### Contributors

- @beenje
- @beeankha
- @chbrandt
- @chenghlee
- @conda-bot
- @dbast
- @dholth
- @duncanmmacleod
- @gforsyth
- @eltociear
- @jaimergp
- @jezdez
- @jmcarpenter2 made their first contribution in https://github.com/conda/conda/pull/13034
- @kenodegard
- @ForgottenProgramme
- @Mon-ius made their first contribution in https://github.com/conda/conda/pull/12811
- @otaithleigh made their first contribution in https://github.com/conda/conda/pull/13035
- @psteyer made their first contribution in https://github.com/conda/conda/pull/11610
- @tarcisioe made their first contribution in https://github.com/conda/conda/pull/9614
- @travishathaway
- @wolfv made their first contribution in https://github.com/conda/conda/pull/13095
- @zeehio made their first contribution in https://github.com/conda/conda/pull/13075
- @pre-commit-ci[bot]

### 4.4.25 23.7.4 (2023-09-12)

#### Enhancements

- Use `os.scandir()` to find conda subcommands without `stat()` overhead. (#13033, #13067)

#### Bug fixes

- Fix S3 bucket name in test suite. (#12989)
- Fix performance regression of basic commands (e.g., `conda info`) on WSL. (#13035)
- Catch `PermissionError` raised by `conda.cli.find_commands.find_commands` when user's `$PATH` contains restricted paths. (#13062, #13089)
- Fix sorting error for `conda config --show-sources --json`. (#13076)

#### Contributors

- @beeankha
- @dholth
- @kenodegard
- @otaithleigh made their first contribution in https://github.com/conda/conda/pull/13035

### 4.4.26 23.7.3 (2023-08-21)

#### Bug fixes

- Fix regression for supporting conda executable plugins installed into non-base environments. (#13006)

#### Contributors

- @kenodegard

### 4.4.27 23.7.2 (2023-07-27)

#### Bug fixes

- Fix regression in parsing `--json` and `--debug` flags for executable plugins. (#12935, #12936)

### Contributors

- @kenodegard

## 4.4.28 23.7.1 (2023-07-26)

### Bug fixes

- Patch parsed args with pre_args to correctly parse `--json` and `--debug` arguments. (#12928, #12929)

### Contributors

- @jezdez
- @kenodegard

## 4.4.29 23.7.0 (2023-07-25)

### Enhancements

- Add `conda.deprecations.DeprecationHandler.action` helper to deprecate `argparse.Action`s. (#12493)
- Add support for the FreeBSD operating system and register `freebsd-64` as a known subdirectory for FreeBSD on x86-64. (#12647)
- Do not mock `$CONDA_PREFIX` when `--name` or `--prefix` is provided. (#12696)
- Add support for `sha256` filters in the MatchSpec syntax (e.g. `*[sha256=f453db4ffe2271ec492a2913af4e61d4a6c118201f07` (#12654 via #12707)
- Add a new health check to `conda doctor` detecting altered packages in an environment by comparing expected and computed `sha256` checksums. (#12757)
- Add new `pre_commands` and `post_commands` plugin hooks allowing plugins to run code before and after `conda` subcommands. (#12712, #12758, #12864)
- Stop using `distutils` directly in favor of the vendored version in `setuptools` 60 and later or standard library equivalents. (#11136)
- Add a `CITATION.cff` file to the root of the repository to make it easier for users to cite conda. (#12781)
- Add optional `CondaSubcommand.configure_parser` allowing third-party plugins to hook into conda's argument parser. (#12814)
- Only display third-party subcommands in `conda --help` and not for every other subcommand. (#12814, #12740)
- Add a new config option, `no_plugins`, a `--no-plugins` command line flag, and a `CONDA_NO_PLUGINS` environment variable that disables external plugins for built-in conda commands. (#12748)
- Register plugins using their canonical/fully-qualified name instead of the easily spoofable entry point name. (#12869)
- De-duplicate plugin and legacy subcommands in `conda --help`. (#12893)
- Implement a 2-phase parser to better handle plugin disabling (via `--no-plugins`). (#12910)
- Refactor subcommand parsing to use a greedy parser since `argparse.REMAINDER` has known issues. (#12910)

### Bug fixes

- Use `requests.exceptions.JSONDecodeError` for ensuring compatibility with different `json` implementations used by requests. This fixes a bug that caused only the first of multiple given source URLs to be tried. This also raises the minimum required requests version to 2.27.0. (#12683)
- Don't export `__osx` virtual package when `CONDA_OVERRIDE_OSX=""`. (#12715)
- Fix erroneous `conda deactivate` behavior of unsetting preexisting environment variables that are identical to those set during `conda activate`. (#12769)
- Correct third-party subcommands to receive *remaining* arguments instead of a blanket `sys.argv[2:]` which broke `conda_cli` testing. (#12814, #12910)

### Deprecations

- Mark `conda.base.context.context.root_dir` as pending deprecation. Use `conda.base.context.context.root_prefix` instead. (#12701)
- Mark `conda.plugins.subcommands.doctor.cli.get_prefix` as pending deprecation. Use `conda.base.context.context.target_prefix` instead. (#12725)
- Mark `conda.models.leased_path_entry.LeasedPathEntry` as pending deprecation. (#12735)
- Mark `conda.models.enums.LeasedPathType` as pending deprecation. (#12735)
- Mark `conda.common.temporary_content_in_file` as pending deprecation. Use `tempfile` instead. (#12795)
- Mark `conda.cli.python_api` as pending deprecation. Use `conda.testing.conda_cli` fixture instead. (#12796)

### Docs

- Document how to use the new `pre_commands` and `post_commands` plugin hooks. (#12712, #12758)
- Add docstrings to all public modules. (#12792)
- Auto-generate API docs using `sphinx-autoapi`. (#12798)
- Convert all manual redirects into config using `sphinx-reredirects`. (#12798)
- Revise the plugins index page to make it easier to understand how to create a conda plugin. (#12802)
- Add missing `conda env` CLI docs. (#12841)

### Other

- Update `tests/cli/test_main_rename.py` to use latest fixtures. (#12517)
- Update `tests/test_activate.py` to test the new behavior. (#12769)
- Re-enable all `conda_env` tests and remove irrelevant tests. (#12813)
- Convert all `unittest`-style tests to `pytest`-style. (#12819)
- Convert `tests/test-recipes` into local noarch packages instead of relying on conda-test channel and local builds. (#12879)

### Contributors

- @beeankha
- @conda-bot
- @dariocurr
- @jaimergp
- @jezdez
- @johanneskoester made their first contribution in https://github.com/conda/conda/pull/12683
- @jjhelmus
- @kalawac made their first contribution in https://github.com/conda/conda/pull/12738
- @kenodegard
- @schackartk made their first contribution in https://github.com/conda/conda/pull/12781
- @lesteve made their first contribution in https://github.com/conda/conda/pull/12715
- @ForgottenProgramme
- @marcoesters made their first contribution in https://github.com/conda/conda/pull/12863
- @mpotane made their first contribution in https://github.com/conda/conda/pull/11740
- @mattkram made their first contribution in https://github.com/conda/conda/pull/12730
- @morremeyer made their first contribution in https://github.com/conda/conda/pull/12871
- @mcg1969
- @travishathaway
- @pre-commit-ci[bot]

## 4.4.30 23.5.2 (2023-07-13)

### Bug fixes

- Correct `native_path_to_unix` failure to handle no paths (e.g., an empty string or an empty iterable). (#12880)

### Contributors

- @kenodegard

## 4.4.31 23.5.1 (2023-07-12)

### Bug fixes

- Add (back) the `cygpath` fallback logic since `cygpath` is not always available on Windows. (#12873)

### Contributors

- @kenodegard

## 4.4.32 23.5.0 (2023-05-17)

### Enhancements

- Add `conda doctor` subcommand plugin. (#474)
- Add Python 3.11 support. (#12256)
- Add `conda list --reverse` to return a reversed list of installed packages. (#11954)
- Switch from `setup.py` to `pyproject.toml` and use Hatchling for our build system. (#12509)
- Optimize which Python modules get imported during `conda activate` calls to make it faster. (#12550)
- Add `conda_cli` fixture to replace `conda.testing.helpers.run_inprocess_conda_command` and `conda.testing.integration.run_command`. (#12592)
- Add `tmp_env` fixture to replace `conda.testing.integration.make_temp_env`. (#12592)
- Add `path_factory` fixture to replace custom prefix logic like `conda.testing.integration._get_temp_prefix` and `conda.testing.integration.make_temp_prefix`. (#12592)
- Refactor the way that the `Activator` classes are defined in `conda/activate.py`. (#12627)
- Warn about misconfiguration when signature verification is enabled. (#12639)

### Bug fixes

- `conda clean` no longer fails if we failed to get the file stats. (#12536)
- Provide fallback version if `conda.deprecations.DeprecationHandler` receives a bad version. (#12541)
- Ensure the default value for `defaults` includes `msys2` when `context.subdir` is `win-*` on non-Windows platforms. (#12555)
- Avoid `TypeError` when non-string types are written to the index cache metadata. (#12562)
- `conda.core.package_cache_data.UrlsData.get_url` no longer fails when `package_path` has `.conda` extension. (#12516)
- Stop pre-converting paths to Unix style on Windows in `conda.sh`, so that they are prefix replaceable upon installation, which got broken by #12509. It also relies on `cygpath` at runtime, which all `msys2`/`cygwin` bash versions on Windows should have available. (#12627)

### Deprecations

- Mark `conda_env.pip_util.get_pip_version` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.PipPackage` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.installed` as pending deprecation. (#12492)
- Mark `conda_env.pip_util._canonicalize_name` as pending deprecation. (#12492)
- Mark `conda_env.pip_util.add_pip_installed` as pending deprecation. (#12492)
- Mark `conda_env.env.load_from_directory` as pending deprecation. (#12492)

- Mark `python -m conda_env.cli.main` as pending deprecation. Use `conda env` instead. (#12492)
- Mark `python -m conda_env` as pending deprecation. Use `conda env` instead. (#12492)
- Mark `conda.auxlib.packaging` for deprecation in 24.3.0. (#12509)
- Rename index cache metadata file `.state.json` to `.info.json` to track draft CEP. (#12669)
- Mark `conda.testing.integration.get_conda_list_tuple` as pending deprecation. Use `conda.core.prefix_data.PrefixData().get()` instead. (#12676)
- Mark `conda.testing.encode_for_env_var` as pending deprecation. (#12677)
- Mark `conda.testing.integration.temp_chdir` as pending deprecation. Use `monkeypatch.chdir` instead. (#12678)

### Docs

- Change the README example from IPython Notebook and NumPy to PyTorch. (#12579)
- Discuss options available to properly configure mirrored channels. (#12583, #12641)
- Add `flake8-docstrings` to `pre-commit`. (#12620)

### Other

- Update retry language in flexible solve and `repodata` logs to be less ominous. (#12612)
- Improve `repodata` / `subdir_data` programming interface (#12521). Index cache metadata has changed to `.info.json` to better align with the draft CEP. Improve cache locking when using `jlap`. Improve `jlap` logging. (#12572)
- Format with `black` and replaced `pre-commit`'s `darker` hook with `black`. (#12554)
- Format with `isort` and add `pre-commit` `isort` hook. (#12554)
- Add functional tests around conda's content trust code. (#11805)
- Enable `flake8` checks that are now handled by `black`. (#12620)

### Contributors

- @beeankha
- @chbrandt made their first contribution in https://github.com/conda/conda/pull/12419
- @chenghlee
- @conda-bot
- @dholth
- @THEdavehogue made their first contribution in https://github.com/conda/conda/pull/12612
- @HeavenEvolved made their first contribution in https://github.com/conda/conda/pull/12496
- @eltociear
- @jaimergp
- @jezdez
- @johnnynunez made their first contribution in https://github.com/conda/conda/pull/12256

- @kenodegard
- @ForgottenProgramme
- @pkmooreanaconda
- @tl-hbk made their first contribution in https://github.com/conda/conda/pull/12604
- @vic-ma made their first contribution in https://github.com/conda/conda/pull/12579
- @pre-commit-ci[bot]
- @sausagenoods made their first contribution in https://github.com/conda/conda/pull/12631

### 4.4.33 23.3.1 (2023-03-28)

#### Enhancements

- Fix and re-enable binstar tests. Replace custom property caching with `functools.cached_property`. (#12495)

#### Bug fixes

- Restore default argument for `SubdirData` method used by `conda-index`. (#12513)
- Include `conda.gateways.repodata.jlap` submodule in package. (#12545)

#### Other

- Add linux-s390x to multi-arch ci/dev container. (#12498)
- Expose a `MINIO_RELEASE` environment variable to provide a way to pin `minio` versions in CI setup scripts. (#12525)
- Add `jsonpatch` dependency to support `--experimental=jlap` feature. (#12528)

#### Contributors

- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @kenodegard
- @ForgottenProgramme

### 4.4.34 23.3.0 (2023-03-14)

#### Enhancements

- Allow the use of environment variables for channel urls in `environment.yaml`. (#10018)
- Improved error message for `conda env create` if the environment file is missing. (#11883)
- Stop using `toolz.dicttoolz.merge` and `toolz.dicttoolz.merge_with`. (#12039)
- Add support for incremental `repodata.json` updates with `--experimental=jlap` on the command line or `experimental: ["jlap"]` in `.condarc` (#12090). Note: switching between "use jlap" and "don't use jlap" invalidates the cache.
- Added a new `conda.deprecations` module for easier & standardized deprecation. Includes decorators to mark functions, modules, classes, and arguments for deprecation and functions to mark modules, constants, and topics for deprecation. (#12125)
- Adds a new `channel_settings` configuration parameter that will be used to override arbitrary settings on per-channel basis. (#12239)
- Improve speed of `repodata.json` parsing by deferring creation of individual `PackageRecord` objects. (#8500)
- Refactor subcommand argument parsing to make it easier to understand. This calls the plugin before invoking the default argument parsing. (#12285)
- Handle I/O errors raised while retrieving channel notices. (#12312)
- Add support for the 64-bit RISC-V architecture on Linux. (#12319)
- Update vendored version of `py-cpuinfo` to 0.9.0. (#12319)
- Improved code coverage. (#12346, #12457, #12469)
- Add a note about `use_only_tar_bz2` being enabled on `PackagesNotFoundError` exceptions. (#12353)
- Added to conda CLI help that `conda remove -n <myenv> --all` can be used to delete environments. (#12378)
- Handle Python import errors gracefully when loading conda plugins from entrypoints. (#12460)

#### Bug fixes

- Fixed errors when renaming without an active environment. (#11915)
- Prevent double solve attempt if `PackagesNotFoundError` is raised. (#12201)
- Virtual packages follow `context.subdir` instead of `platform.system()` to enable cross-platform installations. (#12219)
- Don't export `__glibc` virtual package when `CONDA_OVERRIDE_GLIBC=""`. (#12267)
- Fix `arg_parse` pass-through for `--version` and `--help` in `conda.xsh`. (#12344)
- Filter out `None` path values from `pwd.getpwall()` on Unix systems, for users without home directories, when running as root. (#12063)
- Catch `ChunkedEncodingError` exceptions to prevent network error tracebacks hitting the output. (#12196 via #12487)
- Fix race conditions in `mkdir_p_sudo_safe`. (#12490)

### Deprecations

- Drop `toolz.itertoolz.unique` in favor of custom `conda.common.iterators.unique` implementation. (#12252)
- Stop using `OrderedDict/odict` since `dict` preserves insert order since Python 3.7. (#12254)
- Mark `conda._vendor.boltons` for deprecation in 23.9.0. (#12272, #12482)
- Mark `conda_exe` in `context.py` and a topic in `print_package_info` `cli/main_info.py` for official deprecation. (#12398)
- Remove unused `chain`, `methodcaller`, `mkdtemp`, `StringIO` imports in `conda.common.compat`; apply other fixes from `ruff --fix .` in the test suite. (#12294)
- Remove unused optimization for searching packages based on `*[track_features=<feature name>]`. (#12314)
- Remove Notebook spec support from `conda env`; this was deprecated already and scheduled to be remove in version 4.5. (#12307)
- Mark `conda_exe` in `context.py` and a topic in `print_package_info` `cli/main_info.py` for official deprecation. (#12276)
- Marking `conda.utils.hashsum_file` as pending deprecation. Use `conda.gateways.disk.read.compute_sum` instead. (#12414)
- Marking `conda.utils.md5_file` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12414)
- Marking `conda.gateways.disk.read.compute_md5sum` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "md5")` instead. (#12414)
- Marking `conda.gateways.disk.read.compute_sha256sum` as pending deprecation. Use `conda.gateways.disk.read.compute_sum(path, "sha256")` instead. (#12414)
- Drop Python 3.7 support. (#12436)

### Docs

- Added docs for `conda.deprecations`. (#12452)
- Updated some instances of "Anaconda Cloud" to be "Anaconda.org". (#12238)
- Added documentation on the specifications for `conda search` and `conda install`. (#12304)
- Mark `conda.utils.safe_open` for deprecation. Use builtin `open` instead. (#12415)

### Other

- Update `<cache key>.json.state` `repodata.json` cache format; check `mtime` against cached `repodata.json`. (#12090)
- Skip redundant `tar --no-same-owner` when running as root on Linux, since newer `conda-package-handling` avoids setting ownership from the archive. (#12231)
- Add additional extensions to `conda.common.path` for future use. (#12261)
- Pass `--cov` in test runner scripts but not in `setup.cfg` defaults, for easier debugging. (#12268)
- Constrain conda-build to at least >=3.18.3, released 2019-06-20. (#12309)

- Improve `start.bat` Windows development script. (#12311)
- Provide conda-forge-based Docker images and fix the bundled `minio` binary. (#12335)
- Add support for conda-forge-based CI runtimes. On Linux (all architectures), unit & integration tests will use Python 3.10. On Windows, Python 3.8. On macOS, only the unit tests are run with conda-forge (*instead* of `defaults`!), using Python 3.9. (#12350, #12447 via #12448)
- Fix testing data issue where the `subdir` entry in some files was mismatched. (#12389)
- Initialize conda after installing test requirements during CI. (#12446)
- Speedup pre-commit by a factor of 15 by removing ignored hooks (`pylint/bandit`). This locally reduces the pre-commit runtime from ~43sec to 2.9sec and thus makes it possible to run pre-commit in a loop during development to constantly provide feedback and style the code. (#12466)

### Contributors

- @AdrianFreundQC made their first contribution in https://github.com/conda/conda/pull/11883
- @sanzoghenzo made their first contribution in https://github.com/conda/conda/pull/12074
- @beeankha
- @conda-bot
- @dbast
- @dholth
- @FelisNivalis made their first contribution in https://github.com/conda/conda/pull/11915
- @gforsyth made their first contribution in https://github.com/conda/conda/pull/12344
- @eltociear made their first contribution in https://github.com/conda/conda/pull/12377
- @jaimergp
- @jezdez
- @jjhelmus
- @kannanjayachandran made their first contribution in https://github.com/conda/conda/pull/12363
- @kathatherine
- @kenodegard
- @ForgottenProgramme
- @ryanskeith made their first contribution in https://github.com/conda/conda/pull/12439
- @31Sanskrati made their first contribution in https://github.com/conda/conda/pull/12371
- @travishathaway
- @pre-commit-ci[bot]

### 4.4.35 23.1.0 (2023-01-17)

#### Bug fixes

- Detect CUDA driver version in subprocess. (#11667)
- Fixes the behavior of the `--no-user` flag in `conda init` so that a user's `.bashrc`, etc. remains unaltered, as expected. (#11949)
- Fix several more user facing `MatchSpec` crashes that were identified by fuzzing efforts. (#12099)
- Lock `sys.stdout` to avoid corrupted `--json` multithreaded download progress. (#12231)

#### Docs

- Optional Bash completion support has been removed starting in v4.4.0, and not just deprecated. (#11171)
- Documented optional `channel::package` syntax for specifying dependencies in `environment.yml` files. (#11890)

#### Other

- Refactor `repodata.json` fetching; update on-disk cache format. Based on work by @FFY00. (#11600)
- Environment variable overwriting WARNING is printed only if the env vars are different from those specified in the OS. (#12128)
- Added `conda-libmamba-solver` run constraint. (#12156)
- Updated `ruamel.yaml` version. (#12156)
- Added `tqdm` dependency. (#12191)
- Use `itertools.chain.from_iterable` instead of equivalent `tlz.concat`. (#12165)
- Use `toolz.unique` instead of vendored copy. (#12165)
- Use `itertools.islice` instead of `toolz.take`. (#12165)
- Update CI test workflow to only run test suite when code changes occur. (#12180)
- Added Python 3.10 canary builds. (#12184)

#### Contributors

- @beeankha
- @dholth
- @dariocurr made their first contribution in https://github.com/conda/conda/pull/12128
- @FFY00 made their first contribution in https://github.com/conda/conda/pull/11600
- @jezdez
- @jay-tau made their first contribution in https://github.com/conda/conda/pull/11738
- @kenodegard
- @pkmooreanaconda
- @sven6002 made their first contribution in https://github.com/conda/conda/pull/12162

- @ReveStobinson made their first contribution in https://github.com/conda/conda/pull/12213
- @travishathaway
- @XuehaiPan made their first contribution in https://github.com/conda/conda/pull/11667
- @xylar made their first contribution in https://github.com/conda/conda/pull/11949
- @pre-commit-ci[bot]

### 4.4.36 22.11.1 (2022-12-06)

#### Bug fixes

- Restore default virtual package specs as in 22.9.0 (#12148)
  - re-add `__unix`/`__win` packages
  - restore `__archspec` version/build string composition

#### Other

- Skip test suite for non-code changes. (#12141)

#### Contributors

- @LtDan33
- @jezdez
- @kenodegard
- @mbargull
- @travishathaway

### 4.4.37 22.11.0 (2022-11-23)

#### Enhancements

- Add LD_PRELOAD to env variable list. (#10665)
- Improve CLI warning about updating conda. (#11300)
- Conda's initialize block in the user's profiles will check whether the conda executable exists before calling the conda hook. (#11374)
- Switch to `tqdm` as a real dependency. (#12005)
- Add a new plugin mechanism. (#11435)
- Add an informative message if explicit install fails due to requested packages not being in the cache. (#11591)
- Download and extract packages in parallel. Greatly speeds up package downloads when latency is high. Controlled by the new `fetch_threads` config parameter, defaulting to 5 parallel downloads. Thanks @shuges-uk for reporting. (#11841)
- Add a new plugin hook for virtual packages and convert existing code for virtual packages (from index.py) into plugins. (#11854)

- Require `ruamel.yaml`. (#11868, #11837)
- Stop using `toolz.accumulate`. (#12020)
- Stop using `toolz.groupby`. (#11913)
- Remove vendored `six` package. (#11979)
- Add the ability to extend the solver backends with the `conda_solvers` plugin hook. (#11993)
- Stop using `toolz.functoolz.excepts`. (#12016)
- Stop using `toolz.itertoolz.concatv`. (#12020)
- Also try UTF16 and UTF32 encodings when replacing the prefix. (#9946)

### Bug fixes

- `conda env update` would ask for user input and hang when working with pip installs of git repos and the repo was previously checked out. Tell pip not to ask for user input for that case. (#11818)
- Fix for `conda update` and `conda install` issues related to channel notices. (#11852)
- Signature verification printed `None` when disabled, changes default `metadata_signature_status` to an empty string instead. (#11944)
- Fix importlib warnings when importing conda.cli.python_api on python=3.10. (#11975)
- Several user facing MatchSpec crashes were identified by fuzzing efforts. (#11999)
- Apply minimal fixes to deal with these (and similar) crashes. (#12014)
- Prevent `conda` from using `/bin/sh` + `exec` trick for its own entry point, which drops `$PS1` on some shells (#11885, #11893 via #12043).
- Handle `CTRL+C` during package downloading more gracefully. (#12056)
- Prefer the outer name when a MatchSpec specifies a package's name twice package-name[name=package-name] (#12062)

### Deprecations

- Add a pending deprecation warning for when importing `tqdm` from `conda._vendor`. (#12005)
- Drop `ruamel_yaml` and `ruamel_yaml_conda` in favor of `ruamel.yaml`. (#11837)
- `context.experimental_solver` is now marked for pending deprecation. It is replaced by `context.solver`. The same applies to the `--experimental-solver` flag, the `CONDA_EXPERIMENTAL_SOLVER` environment variable, and the `ExperimentalSolverChoice` enum, which will be replaced by `--solver`, `EXPERIMENTAL_SOLVER` and `SolverChoice`, respectively. (#11889)
- Mark `context.conda_private` as pending deprecation. (#12101)

### Docs

- Add corresponding documentation for the new plugins mechanism. (#11435)
- Update conda cheatsheet for the 4.14.0 release. The cheatsheet now includes an example for `conda rename`. (#11768)
- Document conda-build package format v2, also known as the `.conda`-format. (#11881)
- Remove `allow_other_channels` config option from documentation, as the option no longer exists. (#11866)
- Fix bad URL to "Introduction to conda for Data Scientists" course in conda docs. (#9782)

### Other

- Add a comment to the code that explains why .bashrc is modified on Linux and .bash_profile is modified on Windows/macOS when executing `conda init`. (#11849)
- Add `--mach` and `--arch` options to `dev/start`. (#11851)
- Remove encoding pragma in file headers, as it's not needed in Python 3 anymore. (#11880)
- Refactor `conda init SHELLS` as argparse choices. (#11897)
- Drop pragma fixes from pre-commit checks. (#11909)
- Add pyupgrade to pre-commit checks. This change affects many files. Existing pull requests may need to be updated, rebased, or merged to address conflicts. (#11909)
- Add aarch64 and ppc64le as additional CI platforms for smoke testing. (#11911)
- Serve package files needed for testing using local server. (#12024)
- Update canary builds to guarantee builds for the commits that trigger workflow. (#12040)

### Contributors

- @arq0017 made their first contribution in https://github.com/conda/conda/pull/11810
- @beeankha
- @conda-bot
- @dbast
- @dholth
- @erykoff
- @considerRatio made their first contribution in https://github.com/conda/conda/pull/12028
- @jaimergp
- @jezdez
- @kathatherine
- @kenodegard
- @ForgottenProgramme made their first contribution in https://github.com/conda/conda/pull/11926
- @hmaarrfk made their first contribution in https://github.com/conda/conda/pull/9946
- @NikhilRaverkar made their first contribution in https://github.com/conda/conda/pull/11842

- @pavelzw made their first contribution in https://github.com/conda/conda/pull/11849
- @pkmooreanaconda made their first contribution in https://github.com/conda/conda/pull/12014
- @fragmede made their first contribution in https://github.com/conda/conda/pull/11818
- @SatyamVyas04 made their first contribution in https://github.com/conda/conda/pull/11870
- @timhoffm
- @travishathaway
- @dependabot made their first contribution in https://github.com/conda/conda/pull/11965
- @pre-commit-ci[bot]
- @wulmer

### 4.4.38 22.9.0 (2022-09-14)

#### Special announcement

If you have been following the conda project previously, you will notice a change in our version number for this release. We have officially switched to the CalVer versioning system as agreed upon in CEP 8 (Conda Enhancement Proposal).

Please read that CEP for more information, but here is a quick synopsis. We hope that this versioning system and our release schedule will help make our releases more predictable and transparent to the community going forward. We are now committed to making at least one release every two months, but keep in mind that we can (and most likely will) be making minor version releases within this window.

#### Enhancements

- Replace vendored toolz with toolz dependency. (#11589, #11700)
- Update bundled Python launchers for Windows (`conda/shell/cli-*.exe`) to match the ones found in conda-build. (#11676)
- Add `win-arm64` as a known platform (subdir). (#11778)

#### Bug fixes

- Remove extra prefix injection related to the shell interface breaking `conda run`. (#11666)
- Better support for shebang instructions in prefixes with spaces. (#11676)
- Fix `noarch` entry points in Unicode-containing prefixes on Windows. (#11694)
- Ensure that exceptions that are raised show up properly instead of resulting in a blank `[y/N]` prompt. (#11746)

### Deprecations

- Mark `conda._vendor.toolz` as pending deprecation. (#11704)
- Removes vendored version of urllib3. (#11705)

### Docs

- Added conda capitalization standards to CONTRIBUTING file. (#11712)

### Other

- Add arm64 support to development script `. ./dev/start`. (#11752)
- Update canary-release version to resolve canary build issue. (#11761)
- Renamed canary recipe from `conda.recipe` to `recipe`. (#11774)

### Contributors

- @beeankha
- @chenghlee
- @conda-bot
- @dholth
- @isuruf
- @jaimergp
- @jezdez
- @razzlestorm made their first contribution in https://github.com/conda/conda/pull/11736
- @jakirkham
- @kathatherine
- @kenodegard
- @scdub made their first contribution in https://github.com/conda/conda/pull/11816
- @travishathaway
- @pre-commit-ci[bot]

## 4.4.39 4.14.0 (2022-08-02)

### Enhancements

- Only star activated environment in `conda info --envs`/`conda env list`. (#10764)
- Adds new sub-command, `conda notices`, for retrieving channel notices. (#11462)
- Notices will be intermittently shown after running, `install`, `create`, `update`, `env create` or `env update`. New notices will only be shown once. (#11462)
- Implementation of a new `rename` subcommand. (#11496)

- Split `SSLError` from `HTTPError` to help resolve HTTP 000 errors. (#11564)
- Include the invalid package name in the error message. (#11601)
- Bump requests version (`>=2.20.1`) and drop monkeypatching. (#11643)
- Rename `whitelist_channels` to `allowlist_channels`. (#11647)
- Always mention channel when notifying about a new conda update. (#11671)

### Bug fixes

- Correct a misleading `conda --help` error message. (#11625)
- Fix support for CUDA version detection on WSL2. (#11626)
- Fixed the bug when providing empty environment.yml to `conda env create` command. (#11556, #11630)
- Fix MD5 hash generation for FIPS-enabled systems. (#11658)
- Fixed `TypeError` encountered when logging is set to DEBUG and the package's JSON cannot be read. (#11679)

### Deprecations

- `conda.cli.common.ensure_name_or_prefix` is pending deprecation in a future release. (#11490)
- Mark `conda.lock` as pending deprecation. (#11571)
- Remove lgtm.com config. (#11572)
- Remove Python 2.7 `conda.common.url.is_ipv6_address_win_py27` implementation. (#11573)
- Remove redundant `conda.resolve.dashlist` definition. (#11578)
- Mark `conda_env.cli.common.get_prefix` and `conda.base.context.get_prefix` as pending deprecation in favor of `conda.base.context.determine_target_prefix`. (#11594)
- Mark `conda_env.cli.common.stdout_json` as pending deprecation in favor of `conda.cli.common.stdout_json`. (#11595)
- Mark `conda_env.cli.common.find_prefix_name` as pending deprecation. (#11596)
- Mark `conda.auxlib.decorators.memoize` as pending deprecation in favor of `functools.lru_cache`. (#11597)
- Mark `conda.exports.memoized` as pending deprecation in favor of `functools.lru_cache`. (#11597)
- Mark `conda.exports.handle_proxy_407` as pending deprecation. (#11597)
- Refactor `conda.activate._Activator.get_export_unset_vars` to use `**kwargs` instead of `OrderedDict`. (#11649)
- Mark `conda.another_to_unicode` as pending deprecation. (#11678)

### Docs

- Corresponding documentation of `notices` subcommand. (#11462)
- Corresponding documentation of `rename` subcommand. (#11496)
- Correct docs URL to https://docs.conda.io. (#11508)
- Updated the list of environment variables that can now expand in the Use Condarc section. (#11514)
- Include notice that the "All Users" installation option in the Anaconda Installer is no longer available due to security concerns. (#11528)
- Update `conda-zsh-completeion` link. (#11541)
- Missing `pip` as a dependency when including a pip-installed dependency. (#11543)
- Convert `README.rst` to `README.md`. (#11544)
- Updated docs and CLI help to include information on `conda init` arguments. (#11574)
- Added docs for writing integration tests. (#11604)
- Updated `conda env create` CLI documentation description and examples to be more helpful. (#11611)

### Other

- Display tests summary in CI. (#11558)
- Update `Dockerfile` and `ci-images.yml` flow to build multi arch images. (#11560)
- Rename master branch to main. (#11570)

### Contributors

- @drewja made their first contribution in #11614
- @beeankha
- @topherocity made their first contribution in #11658
- @conda-bot
- @dandv made their first contribution in #11636
- @dbast
- @dholth
- @deepyaman made their first contribution in #11598
- @dogukanteber made their first contribution in #11556/#11630
- @jaimergp
- @kathatherine
- @kenodegard
- @nps1ngh made their first contribution in #10764
- @pseudoyim made their first contribution in #11528
- @SamStudio8 made their first contribution in #11679
- @SamuelWN made their first contribution in #11543

- @spencermathews made their first contribution in #11508
- @timgates42
- @timhoffm made their first contribution in #11601
- @travishathaway
- @esc
- @pre-commit-ci[bot]

### 4.4.40 4.13.0 (2022-05-19)

#### Enhancements

- Introducing `conda clean --logfiles` to remove logfiles generated by conda-libmamba-solver. (#11387)
- Add the solver name and version to the user-agent. (#11415, #11455)
- Attempt parsing HTTP errors as a JSON and extract error details. If present, prefer these details instead of those hard-coded. (#11440)

#### Bug fixes

- Fix inconsistencies with `conda clean --dryrun` (#11385)
- Standardize tarball & package finding in `conda clean` (#11386, #11391)
- Fix `escape_channel_url` logic on Windows (#11416)
- Use 'Accept' header, not 'Content-Type' in GET header (#11446)
- Allow extended user-agent collection to fail but log the exception (#11455)

#### Deprecations

- Removing deprecated `conda.cli.activate`. Originally deprecated in conda 4.6.0 in May 2018. (#11309)
- Removing deprecated `conda.compat`. Originally deprecated in conda 4.6.0 in May 2018. (#11322)
- Removing deprecated `conda.install`. Originally deprecated in conda 4.6.0 in May 2018. (#11323)
- Removing deprecated `conda.cli.main_help`. Originally deprecated in conda 4.6.0 in May 2018. (#11325)
- Removed unused `conda.auxlib.configuration`. (#11349)
- Removed unused `conda.auxlib.crypt`. (#11349)
- Removed unused `conda.auxlib.deprecation`. (#11349)
- Removed unused `conda.auxlib.factory`. (#11349)
- Removed minimally used `conda.auxlib.path`. (#11349)
- Removed `conda.exports.CrossPlatformStLink`, a Windows Python <3.2 fix for `os.lstat.st_nlink`. (#11351)
- Remove Python 2.7 and other legacy code (#11364)
- `conda run --live-stream` aliases `conda run --no-capture-output`. (#11422)
- Removes unused exceptions. (#11424)

- Combines conda_env.exceptions with conda.exceptions. (#11425)
- Drop Python 3.6 support. (#11453)
- Remove outdated test `test_init_dev_and_NoBaseEnvironmentError` (#11469)

### Docs

- Initial implementation of deep dive docs (#11059)
- Correction of RegisterPython description in Windows Installer arguments. (#11312)
- Added autodoc documentation for `conda compare`. (#11336)
- Remove duplicated instruction in manage-python.rst (#11381)
- Updated conda cheatsheet. (#11443)
- Fix typos throughout the codebase (#11448)
- Fix `conda activate` example (#11471)
- Updated `conda` 4.12 cheatsheet with new anaconda distribution version (#11479)

### Other

- Add Python 3.10 as a test target. (#10992)
- Replace custom `conda._vendor` with vendoring (#11290)
- Replace `conda.auxlib.collection.frozendict` with vendored `frozendict` (#11398)
- Reorganize and new tests for `conda.cli.main_clean` (#11360)
- Removing vendored usage of urllib3 and instead implementing our own wrapper around std. lib urllib (#11373)
- Bump vendored `py-cpuinfo` version 4.0.0 → 8.0.0. (#11393)
- Add informational Codecov status checks (#11400)

### Contributors

- @beeankha made their first contribution in #11469
- @ChrisPanopoulos made their first contribution in #11312
- @conda-bot
- @dholth
- @jaimergp
- @jezdez
- @kathatherine made their first contribution in #11443
- @kenodegard
- @kianmeng made their first contribution in #11448
- @simon9500 made their first contribution in #11381
- @thomasrockhu-codecov made their first contribution in #11400
- @travishathaway made their first contribution in #11373

- @pre-commit-ci[bot]

### 4.4.41 4.12.0 (2022-03-08)

#### Enhancements

- Add support for libmamba integrations. (#11193)

  This is a new **experimental and opt-in** feature that allows use of the new conda-libmamba-solver for an improved user experience, based on the libmamba community project - the library version of the mamba package manager.

  Please follow these steps to try out the new libmamba solver integration:

  1. Make sure you have conda-libmamba-solver installed in your conda base environment.
  2. Try out the solver using the `--experimental-solver=libmamba` command line option.

     E.g. with a dry-run to install the `scipy` package:

     ```
     conda create -n demo scipy --dry-run --experimental-solver=libmamba
     ```

     Or install in an activated conda environment:

     ```
     conda activate my-environment
     conda install scipy --experimental-solver=libmamba
     ```

- Make sure that `conda env update -f` sets env vars from the referenced yaml file. (#10652)
- Improve command line argument quoting, especially for `conda run`. (#11189)
- Allow `conda run` to work in read-only environments. (#11215)
- Add support for prelink_message. (#11123)
- Added `conda.CONDA_SOURCE_ROOT`. (#11182)

#### Bug fixes

- Refactored `conda.utils.ensure_comspec_set` into `conda.utils.get_comspec`. (#11168)
- Refactored `conda.cli.common.is_valid_prefix` into `conda.cli.common.validate_prefix`. (#11172)
- Instantiate separate S3 session for thread-safety. (#11038)
- Change overly verbose info log to debug. (#11260)
- Remove five.py and update metaclass definitions. (#11267)
- Remove unnecessary conditional in setup.py (#11013)

### Docs

- Clarify on AIE messaging in download.rst. (#11221)
- Fix conda environment variable echo, update example versions. (#11237)
- Fixed link in docs. (#11268)
- Update profile examples. (#11278)
- Fix typos. (#11070)
- Document conda run command. (#11299)

### Other

- Added macOS to continuous integration. (#10875)
- Added ability to build per-pullrequest review builds. (#11135)
- Improved subprocess handling on Windows. (#11179)
- Add `CONDA_SOURCE_ROOT` env var. (#11182)
- Automatically check copyright/license disclaimer & encoding pragma. (#11183)
- Development environment per Python version. (#11233)
- Add concurrency group to cancel GHA runs on repeated pushes to branch/PR. (#11258)
- Only run GHAs on non-forks. (#11265)

### Contributors

- @opoplawski
- @FaustinCarter
- @jaimergp
- @rhoule-anaconda
- @jezdez
- @hajapy
- @erykoff
- @uwuvalon
- @kenodegard
- @manics
- @NaincyKumariKnoldus
- @autotmp
- @yuvipanda
- @astrojuanlu
- @marcelotrevisani

### 4.4.42 4.11.0 (2021-11-22)

#### Enhancements

- Allow channel_alias to interpolate environment variables.
- Support running conda with PyPy on Windows.
- Add ability to add, append and prepend to sequence values when using the conda config subcommand.
- Support Python 3.10 in version parser.
- Add `XDG_CONFIG_HOME` to the conda search path following the XDG Base Directory Specification (XDGBDS).

#### Bug fixes

- Fix the PowerShell activator to not show an error when unsetting environment variables.
- Remove superfluous `eval` statements in fish shell integration.
- Indent the conda fish integration file using fish_indent.
- Fix handling of environment variables containing equal signs (=).
- Handle permission errors when listing all known prefixes.
- Catch Unicode decoding errors when parsing conda-meta files.
- Fix handling write errors when trying to create package cache or env directories.

#### Docs

- Update path of conda repo in RHEL based systems to `/etc/yum.repos.d/conda.repo`.
- Fix the advanced pip example to stop using the now invalid `file:` prefix.
- Minor docs cleanup and adding Code of Conduct.
- Add auto-built architecture documentation for conda based on the C4 Model. See the conda documentation for more information.
- Expand the contributing documentation with a section about static code analysis and code linting.
- Add developer guide section to the documentation, including a conda architecture overview.
- Stop referring to updating anaconda when `conda update` fails with an error.

#### Other

- Build Docker images periodically on GitHub Actions for the continuous integration testing on Linux, storing them on GitHub Packages's registry for reduced latency and cost when using Docker Hub.
- Simplify the Linux GitHub actions workflows by combining used shell scripts.
- Add periodic GitHub Actions workflow to review old issues in the conda issue tracker and mark them as stale if no feedback is provided in a sensible amount of time, eventually closing them.
- Add periodic GitHub Actions workflow to lock the comment threads of old issues and pull requests in the conda GitHub repository to surface regressions with new issues instead.
- Refactor test suite to use more GitHub Actions runners in parallel, reducing total run time by 50%.

- Switched the issue tracker to use forms with additional questions for bug reporters to help in ticket triage.
- Add and automatically run pre-commit as part of the CI system to improve the code quality continuously and raise issues in contributed patches early on.

  The used code linters are: flake8, pylint and bandit.

  The Python code formatter black is used as well but is only enforced on changed code in a commit and not to the whole code base at once.
- Automatically build the conda package upon the successful merge into the master branch and upload it to the conda-canary channel on anaconda.org.

  To try conda out simply run:

```
conda install -c conda-canary/label/dev conda
```

- Automate adding new issues to public GitHub project board to facilitate issue triage.
- Update GitHub issue and pull request labels to be more consistent.
- Start using rever for release management.
- (preview) Enable one-click gitpod and GitHub Codespaces setup for Linux development.

### Contributors

- Benjamin Bertrand
- Chawye Hsu
- Cheng H. Lee
- Dan Meador
- Daniel Bast
- Daniel Holth
- Gregor Kržmanc
- Hsin-Hsiang Peng
- Ilan Cosman
- Isuru Fernando
- Jaime Rodríguez-Guerra
- Jan-Benedikt Jagusch
- Jannis Leidel
- John Flavin
- Jonas Haag
- Ken Odegard
- Kfir Zvi
- Mervin Fansler
- bfis
- mkincaid

- pre-commit CI

### 4.4.43 4.10.3 (2021-06-29)

#### Bug fixes

- Reverts "Don't create an unused S3 client at import time (#10516)" in 4.10.2 that introduced a regression for users using S3 based channels. (#10756)

### 4.4.44 4.10.2 (2021-06-25)

#### Enhancements

- Add `--dry-run` option to `conda env create` (#10635)
- Print warning about pip-installed dependencies only once (#10638)
- Explicit install now respects `--download-only` flag (#10688)
- Bump vendored tqdm version (#10721)

#### Bug fixes

- Fix `changeps1` handling for PowerShell (#10624)
- Handle unbound `$PS1` so sh activation does not fail with `set -u` (#10701)
- Fix sh activation so `$PATH` is properly restored on errors (#10631)
- Fix `-c` option handling so defaults channel is not always re-added (#10735)
- Fix artifact verification-related warnings and errors (#10627, #10677)
- Fix log level used in `conda/core/prefix_data.py` (#9998)
- Fix log level used when fetching artifact verification metadata (#10621)
- Don't create an unused S3 client at import time (#10516)
- Don't load `binstar_client` until needed (#10692)
- Reflect dropping of older Python versions in `setup.py` (#10642)

#### Docs

- Merge release notes and changelog to reduce maintenance burden (#10745)
- Add mentions to PyPy, Anaconda terms of service (#10329, #10712)
- Update Python versions in examples (#10329, #10744)
- Update install macOS instructions (#10728)

### Contributors

- @AlbertDeFusco
- @awwad
- @casperdcl
- @cgranade
- @chenghlee
- @ColemanTom
- @dan-hook
- @dbast
- @ericpre
- @HedgehogCode
- @jamesp
- @jezdez
- @johnhany97
- @lightmare
- @mattip
- @maxerbubba
- @mrakitin
- @stinos
- @thermokarst

## 4.4.45 4.10.1 (2021-04-12)

### Bug fixes

- Fix version detection for `__linux` virtual package (#10599)
- Fix import from `conda_content_trust` (#10589)
- Fix how URL for verification metadata files are constructed (#10617)
- Partially fix profile `$PATH` setup on MSYS2 (#10459)
- Remove `.empty` directory even when `rsync` is not installed (#10331)

### Contributors

- @awwad
- @chenghlee
- @codepage949
- @niklasholm

## 4.4.46 4.10.0 (2021-03-30)

**NOTE**: This release formally drops support for Python 2.7 and Python < 3.6.

### Enhancements

- Add pilot support for metadata signatures and verification (#10578)
- Add `__linux` virtual package (#10552, #10561)
- Support nested keys when using `conda config --get` (#10447, #10572)
- Support installing default packages when using `conda env create` (#10530)
- Support HTTP sources for `conda env update -f` (#10536)
- Make macOS code signing operations less verbose (#10372)

### Bug fixes

- Fix `conda search` crashing on Python 3.9 (#10542)
- Allow `{channel}::pip` to satisfy pip requirements (#10550)
- Support `{host}:{port}` specifications in environment YAML files (#10417)
- Fall back to system `.condarc` if user `.condarc` is absent (#10479)
- Try UTF-16 if UTF-8 fails when reading environment YAML files (#10356)
- Properly parse Python version >= 3.10 (#10478)
- Fix zsh initialization when `$ZDOTDIR` is defined (#10413)
- Fix path handling for csh (#10410)
- Fix `setup.py` requirement for vendored `ruamel_yaml_conda` (#10441)
- Fix errors when pickling vendored `auxlib` objects (#10386)

### Docs

- Document the `__unix` and `__windows` virtual packages (#10511)
- Update list of supported and default versions of Python (#10531)
- Favor using `pip` instead of `setup.py` when setting up CI (#10308)

### Miscellaneous

- CI: drop Python 2.7 and add Python 3.9 (#10548)

### Contributors

- @awwad
- @BastianZim
- @beenje
- @bgobbi
- @blubs
- @chenghlee
- @cjmartian
- @ericpre
- @erykoff
- @felker
- @giladmaya
- @jamesmyatt
- @mingwandroid
- @opoplawski
- @saadparwaiz1
- @saucoide

## 4.4.47 4.9.2 (2020-11-10)

### Enhancements

- Use vendored `tqdm` in `conda.resolve` for better consistency (#10337)

### Bug fixes

- Revert to previous naming scheme for repodata cache files when `use_only_tar_bz2` config option is false (#10350)

### Docs

- Fix missing release notes (#10342)
- Fix permission errors when configuring deb repositories (#10347)

### Contributors

- @chenghlee
- @csoja
- @dylanmorroll
- @sscherfke

## 4.4.48 4.9.1 (2020-10-26)

### Enhancements

- Respect PEP 440 ~= "compatible release" clause (#10313)

### Bug fixes

- Remove `preload_openssl` for Win32 (#10298)
- Add `if exist` to Windows registry hook (#10305)

### Contributors

- @mingwandroid

## 4.4.49 4.9.0 (2020-10-19)

### Enhancements

- Add `osx-arm64` as a recognized platform (#10128, #10134, #10137)
- Resign files modified during installation on ARM64 macOS (#10260)
- Add `__archspec` virtual package to identify CPU microarchitecture (#9930)
- Add `__unix` and `__win` virtual packages (#10214)
- Add `--no-capture--output` option to `conda run` (#9646)
- Add `--live-stream` option to `conda run` (#10270)
- Export and import environment variables set using `conda env config` (#10169)

- Cache repodata from `file://` channels (#9730)
- Do not relink already-installed packages (#10208)
- Speed up JSON formatting in logz module (#10189)

### Bug fixes:

- Stop `env remove --dry-run` from actually removing environments (#10261)
- Virtual package requirements are now considered by the solver (#10057)
- Fix cached filename processing when using only tar.bz2 (#10193)
- Stop showing solver hints about CUDA when it is not a dependency (#10275)
- Ignore virtual packages when checking environment consistency (#10196)
- Fix `config --json` output errors in certain circumstances (#10194)
- More consistent error handling by `conda shell` (#10238)
- Bump vendored version of tqdm to fix various threading and I/O bugs (#10266)

### Docs

- Correctly state default `/AddToPath` option in Windows installer (#10179)
- Fix typos in `--repodata-fn` help text (#10279)

### Miscellaneous

- Update CI infrastructure to use GitHub Actions (#10176, #10186, #10234)
- Update README badge to show GitHub Actions status (#10254)

### Contributors

- @AlbertDeFusco
- @angloyna
- @bbodenmiller
- @casperdcl
- @chenghlee
- @chrisburr
- @cjmartian
- @dhirschfeld
- @ericpre
- @gabrielcnr
- @InfiniteChai
- @isuruf
- @jjhelmus

- @LorcanHamill
- @maresb
- @mingwandroid
- @mlline00
- @xhochy
- @ydmytryk

### 4.4.50 4.8.5 (2020-09-14)

#### Enhancements

- Add `osx-arm64` as a recognized platform (#10128, #10134)

#### Contributors

- @isuruf
- @jjhelmus

### 4.4.51 4.8.4 (2020-08-06)

#### Enhancements

- Add `linux-ppc64` as a recognized platform (#9797, #9877)
- Add `linux-s390x` as a recognized platform (#9933, #10051)
- Add spinner to pip installer (#10032)
- Add support for running conda in PyPy (#9764)
- Support creating conda environments using remote specification files (#9835)
- Allow request retries on various HTTP errors (#9919)
- Add `compare` command for environments against a specification file (#10022)
- Add (preliminary) support for JSON-format activation (#8727)
- Properly handle the `CURL_CA_BUNDLE` environment variable (#10078)
- More uniformly handle `$CONDA_PREFIX` when exporting environments (#10092)
- Enable trailing _ to anchor OpenSSL-like versions (#9859)
- Replace `listdir` and `glob` with `scandir` (#9889)
- Ignore virtual packages when searching for constrained packages (#10117)
- Add virtual packages to be considered in the solver (#10057)

### Bug fixes:

- Prevent `remove --all` from deleting non-environment directories (#10086)
- Prevent `create --dry-run --yes` from deleting existing environments (#10090)
- Remove extra newline from environment export file (#9649)
- Print help on incomplete `conda env config` command rather than crashing (#9660)
- Correctly set exit code/errorlevel when `conda run` exits (#9665)
- Send "inconsistent environment" warnings to stderr to avoid breaking JSON output (#9738)
- Fix output formatting from post-link scripts (#9841)
- Fix URL parsing for channel subdirs (#9844)
- Fix `conda env export -f` sometimes producing empty output files (#9909)
- Fix handling of Python releases with two-digit minor versions (#9999)
- Do not use gid to determine if user is an admin on *nix platforms (#10002)
- Suppress spurious xonsh activation warnings (#10005)
- Fix crash when running `conda update --all` on a nonexistent environment (#10028)
- Fix collections import for Python 3.8 (#10093)
- Fix regex-related deprecation warnings (#10093, #10096)
- Fix logic error when running under Python 2.7 on 64-bit platforms (#10108)
- Fix Python 3.8 leaked semaphore issue (#10115)

### Docs

- Fix formatting and typos (#9623, #9689, #9898, #10042)
- Correct location for yum repository configuration files (#9988)
- Clarify usage for the `--channel` option (#10054)
- Clarify Python is not installed by default into new environments (#10089)

### Miscellaneous

- Fixes to tests and CI pipelines (#9842, #9863, #9938, #9960, #10010)
- Remove conda-forge dependencies for developing conda (#9857, #9871)
- Audit YAML usage for `safe_load` vs `round_trip_load` (#9902)

### Contributors

- @alanhdu
- @angloyna
- @Anthchirp
- @Arrowbox
- @bbodenmiller
- @beenje
- @bernardoduarte
- @birdsarah
- @bnemanich
- @chenghlee
- @ChihweiLHBird
- @cjmartian
- @ericpre
- @error404-beep
- @esc
- @hartb
- @hugobuddel
- @isuruf
- @jjhelmus
- @kalefranz
- @mingwandroid
- @mlline00
- @mparry
- @mrocklin
- @necaris
- @pdnm
- @pradghos
- @ravigumm
- @Reissner
- @scopatz
- @sidhant007
- @songmeixu
- @speleo3
- @tomsaleeba
- @WinstonPais

### 4.4.52 4.8.3 (2020-03-13)

#### Docs

- Add release notes for 4.8.2 to docs (#9632)
- Fix typos in docs (#9637, #9643)
- Grammatical and formatting changes (#9647)

#### Bug fixes:

- Account for channel is specs (#9748)

#### Contributors

- @bernardoduarte
- @forrestwaters
- @jjhelmus
- @msarahan
- @rrigdon
- @timgates42

### 4.4.53 4.8.2 (2020-01-24)

#### Enhancements

- Solver messaging improvements (#9560)

#### Docs

- Added precedence and conflict info (#9565)
- Added how to set env variables with config API (#9536)
- Updated user guide, deleted Overview, minor clean up (#9581)
- Add code of conduct (#9601, #9602, #9603, #9603, #9604 #9605)

#### Bug fixes:

- change fish prompt only if changeps1 is true (#7000)
- make frozendict JSON serializable (#9539)
- Conda env create empty dir (#9543)

#### Contributors

- @msarahan
- @jjhelmus
- @rrigdon
- @soapy1
- @teake
- @csoja
- @kfranz

### 4.4.54 4.8.1 (2019-12-19)

#### Enhancements

- improve performance for conda run by avoiding Popen.communicate (#9381)
- Put conda keyring in /usr/share/keyrings on Debian (#9424)
- refactor common.logic to fix some bugs and prepare for better modularity (#9427)
- Support nested configuration (#9449)
- Support Object configuration parameters (#9465)
- Use freeze_installed to speed up conda env update (#9511)
- add networking args to conda env create (#9525)

#### Docs

- fix string concatenation running words together regarding CONDA_EXE (#9411)
- Fix typo (“list” -> “info”) (#9433)
- typo in condarc key envs_dirs (#9478)
- Clarify channel priority and package sorting (#9492)
- improve description of DLL loading verification and activating environments (#9453)
- Installing with specific build number (#9534)

#### Bug fixes:

- Fix calling python api run_command with list and string arguments (#9331)
- revert init bash completion (#9421)
- set tmp to shortened path that excludes spaces (#9409)
- avoid function redefinition upon resourcing conda.fish (#9444)
- propagate pip error level when creating envs with conda env (#9460)
- fix incorrect chown call (#9464)
- Add subdir to PackageRecord dist_str (#9418)

- Fix running conda activate in multiple processes on windows (#9477)
- Don't check in pkgs for trash (#9472)
- remove setuptools from run_constrained in recipe (#9485)
- Fix `__conda_activate` function to correctly return exit code (#9532)
- fix overly greedy capture done by subprocess for conda run (#9537)

### Contributors

- @AntoinePrv
- @brettcannon
- @bwildenhain
- @cjmartian
- @felker
- @forrestwaters
- @gilescope
- @isuruf
- @jeremyjliu
- @jjhelmus
- @jhultman
- @marcuscaisey
- @mbargull
- @mingwandroid
- @msarahan
- @okhoma
- @osamoylenko
- @rrigdon
- @ruleofthehuns
- @soapy1
- @tartansandal

## 4.4.55 4.8.0 (2019-11-04)

### Enhancements

- retry downloads if they fail, controlled by `remote_max_retries` and `remote_backoff_factor` configuration values (#9318)
- redact authentication information in some URLs (#9341)
- add osx version virtual package , `__osx` (#9349)
- add glibc virtual package, `__glibc` (#9358)

### Docs

- removeed references to MD5s from docs (#9247)
- Add docs on `CONDA_DLL_SEARCH_MODIFICATION_ENABLED` (#9286)
- document threads, spec history and configuration (#9327)
- more documentation on channels (#9335)
- document the .condarc search order (#9369)
- various minor documentation fixes (#9238, #9248, #9267, #9334, #9351, #9372, #9378, #9388, #9391, #9393)

### Bug fixes

- fix issues with xonsh activation on Windows (#8246)
- remove unsupported --lock argument from conda clean (#8310)
- do not add `sys_prefix_path` to failed activation or deactivation (#9282)
- fix csh setenv command (#9284)
- do not memorize `PackageRecord.combined_depends` (#9289)
- use `CONDA_INTERNAL_OLDPATH` rather than `OLDPATH` in activation script (#9303)
- fixes xonsh activation and tab completion (#9305)
- fix what channels are queried when context.offline is True (#9385)

### Contributors

- @analog-cbarber
- @andreasg123
- @beckermr
- @bryant1410
- @colinbrislawn
- @felker
- @forrestwaters
- @gabrielcnr
- @isuruf
- @jakirkham
- @jeremyjliu
- @jjhelmus
- @jooh
- @jpigla
- @marcelotrevisani
- @melund

- @mfansler
- @mingwandroid
- @msarahan
- @rrigdon
- @scopatz
- @soapy1
- @WillyChen123
- @xhochy

### 4.4.56 4.7.12 (2019-09-12)

#### Enhancements

- add support for env file creation based on explicit specs in history (#9093)
- detect prefix paths when -p nor -n not given (#9135)
- Add config parameter to disable conflict finding (for faster time to errors) (#9190)

#### Bug fixes

- fix race condition with creation of repodata cache dir (#9073)
- fix ProxyError expected arguments (#9123)
- makedirs to initialize .conda folder when registering env - fixes permission errors with .conda folders not existing when package cache gets created (#9215)
- fix list duplicates errors in reading repodata/prefix data (#9132)
- fix neutered specs not being recorded in history, leading to unsatisfiable environments later (#9147)
- Standardize “conda env list” behavior between platforms (#9166)
- add JSON output to conda env create/update (#9204)
- speed up finding conflicting specs (speed regression in 4.7.11) (#9218)

#### Contributors

- @beenje
- @Bezier89
- @cjmartian
- @forrestwaters
- @jjhelmus
- @martin-raden
- @msarahan
- @nganani

- @rrigdon
- @soapy1
- @WesRoach
- @zheaton

## 4.4.57 4.7.11 (2019-08-06)

### Enhancements

- add config for control of number of threads. These can be set in condarc or using environment variables. Names/default values are: `default_threads/None`, `repodata_threads/None`, `verify_threads/1`, `execute_threads/1` (#9044)

### Bug fixes

- fix repodata_fns from condarc not being respected (#8998)
- Fix handling of UpdateModifiers other than FREEZE_INSTALLED (#8999)
- Improve conflict finding graph traversal (#9006)
- Fix setuptools being removed due to conda run_constrains (#9014)
- Avoid calling find_conflicts until all retries are spent (#9015)
- refactor _conda_activate.bat in hopes of improving behavior in parallel environments (#9021)
- Add support for local version specs in PYPI installed packages (#9025)
- fix boto3 initialization race condition (#9037)
- Fix return condition in package_cache_data (#9039)
- utilize libarchive_enabled attribute provided by conda-package-handling to fall back to .tar.bz2 files only. (#9041, #9053)
- Fix menu creation on windows having race condition, leading to popups about python.exe not being found (#9044)
- Improve list error when egg-link leads to extra egg-infos (#9045)
- Fix incorrect RemoveError when operating on an env that has one of conda's deps, but is not the env in which the current conda in use resides (#9054)

### Docs

- Document new package format better
- Document `conda init` command
- Document availability of RSS feed for CDN-backed channels that clone

### Contributors

- @Bezier89
- @forrestwaters
- @hajapy
- @ihnorton
- @matthewwardrop
- @msarahan
- @rogererens
- @rrigdon
- @soapy1

## 4.4.58 4.7.10 (2019-07-19)

### Bug fixes

- fix merging of specs
- fix bugs in building of chains in prefix graph

### Contributors

- @msarahan

## 4.4.59 4.7.9 (2019-07-18)

### Bug fixes

- fix Non records in comprehension
- fix potential keyerror in depth-first search
- fix PackageNotFound attribute error

### Contributors

- @jjhelmus
- @msarahan

### 4.4.60 4.7.8 (2019-07-17)

#### Improvements

- improve unsatisfiable messages - try to group and explain output better. Remove lots of extraneous stuff that was showing up in 4.7.7 (#8910)
- preload openssl on windows to avoid library conflicts and missing library issues (#8949)

#### Bug fixes

- fix handling of channels where more than one channel contains packages with similar name, subdir, version and build_number. This was causing mysterious unsatisfiable errors for some users. (#8938)
- reverse logic check in checking channel equality, because == is not reciprocal to != with py27 (no `__ne__`) (#8938)
- fix an infinite loop or otherwise large process with building the unsatisfiable info. Improve the depth-first search implementation. (#8941)
- streamline fallback paths to unfrozen solve in case frozen fails. (#8942)
- Environment activation output only shows `conda activate envname` now, instead of sometimes showing just `activate`. (#8947)

#### Contributors

- @forrestwaters
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon
- @soapy1

### 4.4.61 4.7.7 (2019-07-12)

#### Improvements

- When an update command doesn't do anything because installed software conflicts with the update, information about the conflict is shown, rather than just saying "all requests are already satisfied" (#8899)

### Bug fixes

- fix missing package_type attr in finding virtual packages (#8917)
- fix parallel operations of loading index to preserve channel ordering (#8921, #8922)
- filter PrefixRecords out from PackageRecords when making a graph to show unsatisfiable deps. Fixes comparison error between mismatched types. (#8924)
- install entry points before running post-link scripts, because post link scripts may depend on entry points. (#8925)

### Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

## 4.4.62 4.7.6 (2019-07-11)

### Improvements

- Improve cuda virtual package conflict messages to show the `__cuda` virtual package as part of the conflict (#8834)
- add additional debugging info to Resolve.solve (#8895)

### Bug fixes

- deduplicate error messages being shown for post-link scripts. Show captured stdout/stderr on failure (#8833)
- fix the checkout step in the Windows dev env setup instructions (#8827)
- bail out early when implicit python pinning renders an explicit spec unsatisfiable (#8834)
- handle edge cases in pinned specs better (#8843)
- extract package again if url is None (#8868)
- update docs regarding indexing and subdirs (#8874)
- remove warning about conda-build needing an update that was bothering people (#8884)
- only add repodata fn into cache key when fn is not repodata.json (#8900)
- allow conda to be downgraded with an explicit spec (#8892)
- add target to specs from historic specs (#8901)
- improve message when solving with a repodata file before repodata.json fails (#8907)
- fix distutils usage for "which" functionality. Fix inability to change python version in envs with noarch packages (#8909)
- fix anaconda metapackage being removed because history matching was too restrictive (#8911)
- make freezing less aggressive; add fallback to non-frozen solve (#8912)

### Contributors

- @forrestwaters
- @jjhelmus
- @mcopes73
- @msarahan
- @richardjgowers
- @rrigdon
- @soapy1
- @twinssbc

## 4.4.63 4.7.5 (2019-06-24)

### Improvements

- improve wording in informational message when a particular `*_repodata.json` can't be found. No need for alarm. (#8808)

### Bug fixes

- restore tests being run on win-32 appveyor (#8801)
- fix Dist class handling of .conda files (#8816)
- fix strict channel priority handling when a package is unsatisfiable and thus not present in the collection (#8819)
- handle JSONDecodeError better when package is corrupted at extract time (#8820)

### Contributors

- @dhirschfeld
- @msarahan
- @rrigdon

## 4.4.64 4.7.4 (2019-06-19)

### Improvements

- Revert to and improve the unsatisfiability determination from 4.7.2 that was reverted in 4.7.3. It's faster. (#8783)

### Bug fixes

- fix tcsh/csh init scripts (#8792)

### Docs improvements

- clean up docs of run_command
- fix broken links
- update docs environment.yaml file to update conda-package-handling
- conda logo favicon
- update strict channel priority info
- noarch package content ported from conda-forge
- add info about conda-forge
- remove references to things as they were before conda 4.1. That was a long time ago. This is not a history book.

### Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

## 4.4.65 4.7.3 (2019-06-14)

### Bug fixes

- target prefix overrid applies to entry points in addition to replacements in standard files (#8769)
- Revert to solver-based unsatisfiability determination (#8775)
- fix renaming of existing prompt function in powershell (#8774)

### Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @ScottEvtuch

### 4.4.66 4.7.2 (2019-06-10)

#### Behavior changes

- unsatisfiability is determined in a slightly different way now. It no longer uses the SAT solver, but rather determines whether any specs have no candidates at all after running through get_reduced_index. This has been faster in benchmarks, but we welcome further data from your use cases about whether this was a good change. (#8741)
- when using the --only-deps flag for the `install` command, conda now explicitly records those specs in your history. This primarily serves to reduce conda accidentally removing packages that you have actually requested. (#8766)

#### Improvements

- UnsatisfiableError messages are now grouped into categories and explained a bit better. (#8741)
- –repodata-fn argument can be passed multiple times to have more fallback paths. `repodata_fns` conda config setting does the same thing, but saves you from needing to do it for every command invocation. (#8741)

#### Bug fixes

- fix channel flip-flopping that was happening when adding a channel other than earlier ones (#8741)
- refactor flow control for multiple repodata files to not use exceptions (#8741)
- force conda to use only old .tar.bz2 files if conda-build <3.18.3 is installed. Conda-build breaks when inspecting file contents, and this is fixed in conda-build 3.18.3 (#8741)
- use --force when using rsync to improve behavior with folders that may exist in the destination somehow. (#8750)
- handle EPERM errors when renaming, because MacOS lets you remove or create files, but not rename them. Thanks Apple. (#8755)
- fix conda removing packages installed via `install` with --only-deps flag when either `update` or `remove` commands are run. See behavior changes above. (#8766)

#### Contributors

- @csosborn
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon

### 4.4.67 4.7.1 (2019-05-30)

#### Improvements

- Base initial solver specs map on explicitly requested specs (new and historic) (#8689)
- Improve anonymization of automatic error reporting (#8715)
- Add option to keep using .tar.bz2 files, in case new .conda isn't working for whatever reason (#8723)

#### Bug fixes

- fix parsing hyphenated PyPI specs (change hyphens in versions to .) (#8688)
- fix PrefixRecord creation when file inputs are .conda files (#8689)
- fix PrefixRecord creation for pip-installed packages (#8689)
- fix progress bar stopping at 75% (no extract progress with new libarchive) (#8689)
- preserve pre-4.7 download() interface in conda.exports (#8698)
- virtual packages (such as cuda) are represented by leading double underscores by convention, to avoid confusion with existing single underscore packages that serve other purposes (#8738)

#### Deprecations/Breaking Changes

- The `--prune` flag no longer does anything. Pruning is implicitly the standard behavior now as a result of the initial solver specs coming from explicitly requested specs. Conda will remove packages that are not explicitly requested and are not required directly or indirectly by any explicitly installed package.

#### Docs improvements

- Document removal of the `free` channel from defaults (#8682)
- Add reference to conda config --describe (#8712)
- Add a tutorial for .condarc modification (#8737)

#### Contributors

- @alexhall
- @cjmartian
- @kalefranz
- @martinkou
- @msarahan
- @rrigdon
- @soapy1

### 4.4.68 4.7.0 (2019-05-17)

#### Improvements

- Implement support for "virtual" CUDA packages, to make conda consider the system-installed CUDA driver and act accordingly (#8267)
- Support and prefer new .conda file format where available (#8265, #8639)
- Use comma-separated env names in prompt when stacking envs (#8431)
- show valid choices in error messages for enums (#8602)
- freeze already-installed packages when running `conda install` as a first attempt, to speed up the solve in existing envs. Fall back to full solve as necessary (#8260, #8626)
- add optimization criterion to prefer arch over noarch packages when otherwise equivalent (#8267)
- Remove `free` channel from defaults collection. Add `restore_free_channel` config parameter if you want to keep it. (#8579)
- Improve unsatisfiable hints (#8638)
- Add capability to use custom repodata filename, for smaller subsets of repodata (#8670)
- Parallelize SubdirData readup (#8670)
- Parallelize transaction verification and execution (#8670)

#### Bug fixes

- Fix PATH handling with deactivate.d scripts (#8464)
- Fix usage of deprecated collections ABCs (#)
- fix tcsh/csh initialization block (#8591)
- fix missing CWD display in powershell prompt (#8596)
- `wrap_subprocess_call`: fallback to sh if no bash (#8611)
- move `TemporaryDirectory` to avoid importing from `conda.compat` (#8671)
- fix missing conda-package-handling dependency in dev/start (#8624)
- fix `path_to_url` string index out of range error (#8265)
- fix conda init for xonsh (#8644)
- fix fish activation (#8645)
- improve error handling for read-only filesystems (#8665, #8674)
- break out of minimization when bisection has nowhere to go (#8672)
- Handle None values for link channel name gracefully (#8680)

### Contributors

- @chrisburr
- @EternalPhane
- @jjhelmus
- @kalefranz
- @mbargull
- @msarahan
- @rrigdon
- @scopatz
- @seibert
- @soapy1
- @nehaljwani
- @nh3
- @teake
- @yuvalreches

## 4.4.69 4.6.14 (2019-04-17)

### Bug fixes

- export extra function in powershell Conda.psm1 script (fixes anaconda powershell prompt) (#8570)

### Contributors

- @msarahan

## 4.4.70 4.6.13 (2019-04-16)

### Bug fixes

- disable `test_legacy_repodata` on win-32 (missing dependencies) (#8540)
- Fix activation problems on windows with bash, powershell, and batch. Improve tests. (#8550, #8564)
- pass -U flag to for pip dependencies in conda env when running "conda env update" (#8542)
- rename `conda.common.os` to `conda.common._os` to avoid shadowing os built-in (#8548)
- raise exception when pip subprocess fails with conda env (#8562)
- fix installing recursive requirements.txt files in conda env specs with python 2.7 (#8562)
- Don't modify powershell prompt when "changeps1" setting in condarc is False (#8465)

### Contributors

- @dennispg
- @jjhelmus
- @jpgill86
- @mingwandroid
- @msarahan
- @noahp

## 4.4.71 4.6.12 (2019-04-10)

### Bug fixes

- Fix compat import warning (#8507)
- Adjust collections import to avoid deprecation warning (#8499)
- Fix bug in CLI tests (#8468)
- Disallow the number sign in environment names (#8521)
- Workaround issues with noarch on certain repositories (#8523)
- Fix activation on Windows when spaces are in path (#8503)
- Fix conda init profile modification for powershell (#8531)
- Point conda.bat to condabin (#8517)
- Fix various bugs in activation (#8520, #8528)

### Docs improvements

- Fix links in README (#8482)
- Changelogs for 4.6.10 and 4.6.11 (#8502)

### Contributors

@Bezier89 @duncanmmacleod @ivigamberdiev @javabrett @jjhelmus @katietz @mingwandroid @msarahan @nehaljwani @rrigdon

## 4.4.72 4.6.11 (2019-04-04)

### Bug fixes

- Remove sys.prefix from front of PATH in basic_posix (#8491)
- add import to fix conda.core.index.get_index (#8495)

### Docs improvements

- Changelogs for 4.6.10

### Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

## 4.4.73 4.6.10 (2019-04-01)

### Bug fixes

- Fix python-3 only FileNotFoundError usage in initialize.py (#8470)
- Fix more JSON encode errors for the _Null data type (#8471)
- Fix non-posix-compliant == in conda.sh (#8475, #8476)
- improve detection of pip dependency in environment.yml files to avoid warning message (#8478)
- fix condabin\conda.bat use of dp0, making PATH additions incorrect (#8480)
- init_fish_user: don't assume config file exists (#8481)
- Fix for chcp output ending with . (#8484)

### Docs improvements

- Changelogs for 4.6.8, 4.6.9

### Contributors

- @duncanmmacleod
- @nehaljwani
- @ilango100
- @jjhelmus
- @mingwandroid
- @msarahan
- @rrigdon

### 4.4.74 4.6.9 (2019-03-29)

#### Improvements

- Improve CI for docs commits (#8387, #8401, #8417)
- Implement `conda init --reverse` to undo rc file and registry changes (#8400)
- Improve handling of unicode systems (#8342, #8435)
- Force the "COMSPEC" environment variable to always point to cmd.exe on windows. This was an implicit assumption that was not always true. (#8457, #8461)

#### Bug fixes

- Add central C:/ProgramData/conda as a search path on Windows (#8272)
- remove direct use of ruamel_yaml (prefer internal abstraction, yaml_load) (#8392)
- Fix/improve `conda init` support for fish shell (#8437)
- Improve solver behavior in the presence of inconsistent environments (such as pip as a conda dependency of python, but also installed via pip itself) (#8444)
- Handle read-only filesystems for environments.txt (#8451, #8453)
- Fix conda env commands involving pip-installed dependencies being installed into incorrect locations (#8435)

#### Docs improvements

- updated cheatsheet (#8402)
- updated color theme (#8403)

#### Contributors

- @blackgear
- @dhirschfeld
- @jakirkham
- @jjhelmus
- @katietz
- @mingwandroid
- @msarahan
- @nehaljwani
- @rrigdon
- @soapy1
- @spamlrot-tic

### 4.4.75 4.6.8 (2019-03-06)

#### Bug fixes

- detect when parser fails to parse arguments (#8328)
- separate post-link script running from package linking. Do linking of all packages first, then run any post-link scripts after all packages are present. Ideally, more forgiving in presence of cycles. (#8350)
- quote path to temporary requirements files generated by conda env. Fixes issues with spaces. (#8352)
- improve some exception handling around checking for presence of folders in extraction of tarballs (#8360)
- fix reporting of packages when channel name is None (#8379)
- fix the post-creation helper message from "source activate" to "conda activate" (#8370)
- Add safety checks for directory traversal exploits in tarfiles. These may be disabled using the `safety_checks` configuration parameter. (#8374)

#### Docs improvements

- document MKL DLL hell and new Python env vars to control DLL search behavior (#8315)
- add github template for reporting speed issues (#8344)
- add in better use of sphinx admonitions (notes, warnings) for better accentuation in docs (#8348)
- improve skipping CI builds when only docs changes are involved (#8336)

#### Contributors

- @albertmichaelj
- @jjhelmus
- @matta9001
- @msarahan
- @rrigdon
- @soapy1
- @steffenvan

### 4.4.76 4.6.7 (2019-02-21)

#### Bug fixes

- skip scanning folders for contents during reversal of transactions. Just ignore folders. A bit messier, but a lot faster. (#8266)
- fix some logic in renaming trash files to fix permission errors (#8300)
- wrap pip subprocess calls in conda-env more cleanly and uniformly (#8307)
- revert conda prepending to PATH in cli main file on windows (#8307)
- simplify `conda run` code to use activation subprocess wrapper. Fix a few conda tests to use `conda run`. (#8307)

### Docs improvements

- fixed duplicated “to” in managing envs section (#8298)
- flesh out docs on activation (#8314)
- correct git syntax for adding a remote in dev docs (#8316)
- unpin sphinx version in docs requirements (#8317)

### Contributors

- @jjhelmus
- @MarckK
- @msarahan
- @rrigdon
- @samgd

## 4.4.77 4.6.6 (2019-02-18)

### Bug fixes

- fix incorrect syntax prepending to PATH for conda CLI functionality (#8295)
- fix rename_tmp.bat operating on folders, leading to hung interactive dialogs. Operate only on files. (#8295)

### Contributors

- @mingwandroid
- @msarahan

## 4.4.78 4.6.5 (2019-02-15)

### Bug fixes

- Make super in resolve.py python 2 friendly (#8280)
- support unicode paths better in activation scripts on Windows (#)
- set PATH for conda.bat to include Conda’s root prefix, so that libraries can be found when using conda when the root env is not activated (#8287, #8292)
- clean up warnings/errors about rsync and trash files (#8290)

#### Contributors

- @jjhelmus
- @mingwandroid
- @msarahan
- @rrigdon

### 4.4.79 4.6.4 (2019-02-13)

#### Improvements

- allow configuring location of instrumentation records (#7849)
- prepend conda-env pip commands with env activation to fix library loading (#8263)

#### Bug fixes

- resolve #8176 SAT solver choice error handling (#8248)
- document `pip_interop_enabled` config parameter (#8250)
- ensure prefix temp files are inside prefix (#8253)
- ensure `script_caller` is bound before use (#8254)
- fix overzealous removal of folders after cleanup of failed post-link scripts (#8259)
- fix #8264: Allow 'int' datatype for values to non-sequence parameters (#8268)

#### Deprecations/Breaking Changes

- remove experimental `featureless_minimization_disabled` feature flag (#8249)

#### Contributors

- @davemasino
- @geremih
- @jjhelmus
- @kalefranz
- @msarahan
- @minrk
- @nehaljwani
- @prusse-martin
- @rrigdon
- @soapy1

### 4.4.80 4.6.3 (2019-02-07)

#### Improvements

- Implement `-stack` switch for powershell usage of conda (#8217)
- Enable system-wide initialization for conda shell support (#8219)
- Activate environments prior to running post-link scripts (#8229)
- Instrument more solve calls to prioritize future optimization efforts (#8231)
- print more env info when searching in envs (#8240)

#### Bug fixes

- resolve #8178, fix conda pip interop assertion error with egg folders (#8184)
- resolve #8157, fix token leakage in errors and config output (#8163)
- resolve #8185, fix conda package filtering with embedded/vendored python metadata (#8198)
- resolve #8199, fix errors on .* in version specs that should have been specific to the ~= operator (#8208)
- fix .bat scripts for handling paths on Windows with spaces (#8215)
- fix powershell scripts for handling paths on Windows with spaces (#8222)
- handle missing rename script more gracefully (especially when updating/installing conda itself) (#8212)

#### Contributors

- @dhirschfeld
- @jjhelmus
- @kalefranz
- @msarahan
- @murrayreadccdc
- @nehaljwani
- @rrigdon
- @soapy1

### 4.4.81 4.6.2 (2019-01-29)

#### Improvements

- Documentation restructuring/improvements (#8139, #8143)
- rewrite rm_rf to use native system utilities and rename trash files (#8134)

### Bug fixes

- fix UnavailableInvalidChannel errors when only noarch subdir is present (#8154)
- document, but disable the `allow_conda_downgrades` flag, pending re-examination of the warning, which was blocking conda operations after an upgrade-downgrade cycle across minor versions. (#8160)
- fix conda env export missing pip entries without use of pip interop enabled setting (#8165)

### Contributors

- @jjhelmus
- @msarahan
- @nehaljwani
- @rrigdon

## 4.4.82 4.5.13 (2019-01-29)

### Improvements

- document the allow_conda_downgrades configuration parameter (#8034)
- remove conda upgrade message (#8161)

### Contributors

- @msarahan
- @nehaljwani

## 4.4.83 4.6.1 (2019-01-21)

### Improvements

- optimizations in `get_reduced_index` (#8117, #8121, #8122)

### Bug Fixes

- fix faulty onerror call for rm (#8053)
- fix activate.bat to use more direct call to conda.bat (don't require conda init; fix non-interactive script) (#8113)

#### Contributors

- @jjhelmus
- @msarahan
- @pv

### 4.4.84 4.6.0 (2019-01-15)

#### New Feature Highlights

- resolve #7053 preview support for conda operability with pip; disabled by default (#7067, #7370, #7710, #8050)
- conda initialize (#6518, #7388, #7629)
- resolve #7194 add '–stack' flag to 'conda activate'; remove max_shlvl config (#7195, #7226, #7233)
- resolve #7087 add non-conda-installed python packages into PrefixData (#7067, #7370)
- resolve #2682 add 'conda run' preview support (#7320, #7625)
- resolve #626 conda wrapper for PowerShell (#7794, #7829)

#### Deprecations/Breaking Changes

- resolve #6915 remove 'conda env attach' and 'conda env upload' (#6916)
- resolve #7061 remove pkgs/pro from defaults (#7162)
- resolve #7078 add deprecation warnings for 'conda.cli.activate', 'conda.compat', and 'conda.install' (#7079)
- resolve #7194 add '–stack' flag to 'conda activate'; remove max_shlvl config (#7195)
- resolve #6979, #7086 remove Dist from majority of project (#7216, #7252)
- fix #7362 remove --license from conda info and related code paths (#7386)
- resolve #7309 deprecate 'conda info package_name' (#7310)
- remove 'conda clean --source-cache' and defer to conda-build (#7731)
- resolve #7724 move windows package cache and envs dirs back to .conda directory (#7725)
- disallow env names with colons (#7801)

#### Improvements

- import speedups (#7122)
- –help cleanup (#7120)
- fish autocompletion for conda env (#7101)
- remove reference to 'system' channel (#7163)
- add http error body to debug information (#7160)
- warn creating env name with space is not supported (#7168)
- support complete MatchSpec syntax in environment.yml files (#7178)
- resolve #4274 add option to remove an existing environment with 'conda create' (#7133)

- add ability for conda prompt customization via 'env_prompt' config param (#7047)
- resolve #7063 add license and license_family to MatchSpec for 'conda search' (#7064)
- resolve #7189 progress bar formatting improvement (#7191)
- raise log level for errors to error (#7229)
- add to conda.exports (#7217)
- resolve #6845 add option -S / --satisfied-skip-solve to exit early for satisfied specs (#7291)
- add NoBaseEnvironmentError and DirectoryNotACondaEnvironmentError (#7378)
- replace menuinst subprocessing by ctypes win elevation (4.6.0a3) (#7426)
- bump minimum requests version to stable, unbundled release (#7528)
- resolve #7591 updates and improvements from namespace PR for 4.6 (#7599)
- resolve #7592 compatibility shims (#7606)
- user-agent context refactor (#7630)
- solver performance improvements with benchmarks in common.logic (#7676)
- enable fuzzy-not-equal version constraint for pip interop (#7711)
- add -d short option for --dry-run (#7719)
- add --force-pkgs-dirs option to conda clean (#7719)
- address #7709 ensure --update-deps unlocks specs from previous user requests (#7719)
- add package timestamp information to output of 'conda search --info' (#7722)
- resolve #7336 'conda search' tries "fuzzy match" before showing PackagesNotFound (#7722)
- resolve #7656 strict channel priority via 'channel_priority' config option or --strict-channel-priority CLI flag (#7729)
- performance improvement to cache **hash** value on PackageRecord (#7715)
- resolve #7764 change name of 'condacmd' dir to 'condabin'; use on all platforms (#7773)
- resolve #7782 implement PEP-440 '~=' compatible release operator (#7783)
- disable timestamp prioritization when not needed (#7894, #8012)
- compile pyc files for noarch packages in batches (#8015)
- disable per-file sha256 safety checks by default; add extra_safety_checks condarc option to enable them (#8017)
- shorten retries for file removal on windows, where in-use files can't be removed (#8024)
- expand env vars in `custom_channels`, `custom_multichannels`, `default_channels`, `migrated_custom_channels`, and `whitelist_channels` (#7826)
- encode repodata to utf-8 while caching, to fix unicode characters in repodata (#7873)

### Bug Fixes

- fix #7107 verify hangs when a package is corrupted (#7131)
- fix #7145 progress bar uses stderr instead of stdout (#7146)
- fix typo in conda.fish (#7152)
- fix #2154 conda remove should complain if requested removals don't exist (#7135)
- fix #7094 exit early for --dry-run with explicit and clone (#7096)
- fix activation script sort order (#7176)
- fix #7109 incorrect chown with sudo (#7180)
- fix #7210 add suppressed --mkdir back to 'conda create' (fix for 4.6.0a1) (#7211)
- fix #5681 conda env create / update when --file does not exist (#7385)
- resolve #7375 enable conda config --set update_modifier (#7377)
- fix #5885 improve conda env error messages and add extra tests (#7395)
- msys2 path conversion (#7389)
- fix autocompletion in fish (#7575)
- fix #3982 following 4.4 activation refactor (#7607)
- fix #7242 configuration load error message (#7243)
- fix conda env compatibility with pip 18 (#7612)
- fix #7184 remove conflicting specs to find solution to user's active request (#7719)
- fix #7706 add condacmd dir to cmd.exe path on first activation (#7735)
- fix #7761 spec handling errors in 4.6.0b0 (#7780)
- fix #7770 'conda list regex' only applies regex to package name (#7784)
- fix #8076 load metadata from index to resolve inconsistent envs (#8083)

### Non-User-Facing Changes

- resolve #6595 use OO inheritance in activate.py (#7049)
- resolve #7220 pep8 project renamed to pycodestyle (#7221)
- proxy test routine (#7308)
- add .mailmap and .cla-signers (#7361)
- add copyright headers (#7367)
- rename common.platform to common.os and split among windows, linux, and unix utils (#7396)
- fix windows test failures when symlink not available (#7369)
- test building conda using conda-build (#7251)
- solver test metadata updates (#7664)
- explicitly add Mapping, Sequence to common.compat (#7677)
- add debug messages to communicate solver stages (#7803)
- add undocumented sat_solver config parameter (#7811)

### Preview Releases

- 4.6.0a1 at d5bec21d1f64c3bc66c2999cfc690681e9c46177 on 2018-04-20
- 4.6.0a2 at c467517ca652371ebc4224f0d49315b7ec225108 on 2018-05-01
- 4.6.0b0 at 21a24f02b2687d0895de04664a4ec23ccc75c33a on 2018-09-07
- 4.6.0b1 at 1471f043eed980d62f46944e223f0add6a9a790b on 2018-10-22
- 4.6.0rc1 at 64bde065f8343276f168d2034201115dff7c5753 on 2018-12-31

### Contributors

- @cgranade
- @fabioz
- @geremih
- @goanpeca
- @jesse-
- @jjhelmus
- @kalefranz
- @makbigc
- @mandeep
- @mbargull
- @msarahan
- @nehaljwani
- @ohadravid
- @teake

## 4.4.85 4.5.12 (2018-12-10)

### Improvements

- backport 'allow_conda_downgrade' configuration parameter, default is False (#7998)
- speed up verification by disabling per-file sha256 checks (#8017)
- indicate Python 3.7 support in setup.py file (#8018)
- speed up solver by reduce the size of reduced index (#8016)
- speed up solver by skipping timestamp minimization when not needed (#8012)
- compile pyc files more efficiently, will speed up install of noarch packages (#8025)
- avoid waiting for removal of files on Windows when possible (#8024)

### Bug Fixes

- update integration tests for removal of 'features' key (#7726)
- fix conda.bat return code (#7944)
- ensure channel name is not NoneType (#8021)

### Contributors

- @debionne
- @jjhelmus
- @kalefranz
- @msarahan
- @nehaljwani

## 4.4.86 4.5.11 (2018-08-21)

### Improvements

- resolve #7672 compatibility with ruamel.yaml 0.15.54 (#7675)

### Contributors

- @CJ-Wright
- @mbargull

## 4.4.87 4.5.10 (2018-08-13)

### Bug Fixes

- fix conda env compatibility with pip 18 (#7627)
- fix py37 compat 4.5.x (#7641)
- fix #7451 don't print name, version, and size if unknown (#7648)
- replace glob with fnmatch in PrefixData (#7645)

### Contributors

- @jesse-
- @nehaljwani

### 4.4.88 4.5.9 (2018-07-30)

#### Improvements

- resolve #7522 prevent conda from scheduling downgrades (#7598)
- allow skipping feature maximization in resolver (#7601)

#### Bug Fixes

- fix #7559 symlink stat in localfs adapter (#7561)
- fix #7486 activate with no PATH set (#7562)
- resolve #7522 prevent conda from scheduling downgrades (#7598)

#### Contributors

- @kalefranz
- @loriab

### 4.4.89 4.5.8 (2018-07-10)

#### Bug Fixes

- fix #7524 should_bypass_proxies for requests 2.13.0 and earlier (#7525)

#### Contributors

- @kalefranz

### 4.4.90 4.5.7 (2018-07-09)

#### Improvements

- resolve #7423 add upgrade error for unsupported repodata_version (#7415)
- raise CondaUpgradeError for conda version downgrades on environments (#7517)

#### Bug Fixes

- fix #7505 temp directory for UnlinkLinkTransaction should be in target prefix (#7516)
- fix #7506 requests monkeypatch fallback for old requests versions (#7515)

#### Contributors

- @kalefranz
- @nehaljwani

### 4.4.91 4.5.6 (2018-07-06)

#### Bug Fixes

- resolve #7473 py37 support (#7499)
- fix #7494 History spec parsing edge cases (#7500)
- fix requests 2.19 incompatibility with NO_PROXY env var (#7498)
- resolve #7372 disable http error uploads and CI cleanup (#7498, #7501)

#### Contributors

- @kalefranz

### 4.4.92 4.5.5 (2018-06-29)

#### Bug Fixes

- fix #7165 conda version check should be restricted to channel conda is from (#7289, #7303)
- fix #7341 ValueError n cannot be negative (#7360)
- fix #6691 fix history file parsing containing comma-joined version specs (#7418)
- fix msys2 path conversion (#7471)

#### Contributors

- @goanpeca
- @kalefranz
- @mingwandroid
- @mbargull

### 4.4.93 4.5.4 (2018-05-14)

#### Improvements

- resolve #7189 progress bar improvement (#7191 via #7274)

### Bug Fixes

- fix twofold tarball extraction, improve progress update (#7275)
- fix #7253 always respect copy LinkType (#7269)

### Contributors

- @jakirkham
- @kalefranz
- @mbargull

## 4.4.94 4.5.3 (2018-05-07)

### Bug Fixes

- fix #7240 conda's configuration context is not initialized in conda.exports (#7244)

## 4.4.95 4.5.2 (2018-04-27)

### Bug Fixes

- fix #7107 verify hangs when a package is corrupted (#7223)
- fix #7094 exit early for --dry-run with explicit and clone (#7224)
- fix activation/deactivation script sort order (#7225)

## 4.4.96 4.5.1 (2018-04-13)

### Improvements

- resolve #7075 add anaconda.org search message to PackagesNotFoundError (#7076)
- add CondaError details to auto-upload reports (#7060)

### Bug Fixes

- fix #6703,#6981 index out of bound when running deactivate on fish shell (#6993)
- properly close over $_CONDA_EXE variable (#7004)
- fix condarc map parsing with comments (#7021)
- fix #6919 csh prompt (#7041)
- add _file_created attribute (#7054)
- fix handling of non-ascii characters in custom_multichannels (#7050)
- fix #6877 handle non-zero return in CSH (#7042)
- fix #7040 update tqdm to version 4.22.0 (#7157)

### 4.4.97 4.5.0 (2018-03-20)

#### New Feature Highlights

- A new flag, '–envs', has been added to 'conda search'. In this mode, 'conda search' will look for the package query in existing conda environments on your system. If ran as UID 0 (i.e. root) on unix systems or as an Administrator user on Windows, all known conda environments for all users on the system will be searched. For example, 'conda search --envs openssl' will show the openssl version and environment location for all conda-installed openssl packages.

#### Deprecations/Breaking Changes

- resolve #6886 transition defaults from repo.continuum.io to repo.anaconda.com (#6887)
- resolve #6192 deprecate 'conda help' in favor of --help CLI flag (#6918)
- resolve #6894 add http errors to auto-uploaded error reports (#6895)

#### Improvements

- resolve #6791 conda search --envs (#6794)
- preserve exit status in fish shell (#6760)
- resolve #6810 add CONDA_EXE environment variable to activate (#6923)
- resolve #6695 outdated conda warning respects --quiet flag (#6935)
- add instructions to activate default environment (#6944)

#### API

- resolve #5610 add PrefixData, SubdirData, and PackageCacheData to conda/api.py (#6922)

#### Bug Fixes

- channel matchspec fixes (#6893)
- fix #6930 add missing return statement to S3Adapter (#6931)
- fix #5802, #6736 enforce disallowed_packages configuration parameter (#6932)
- fix #6860 infinite recursion in resolve.py for empty track_features (#6928)
- set encoding for PY2 stdout/stderr (#6951)
- fix #6821 non-deterministic behavior from MatchSpec merge clobbering (#6956)
- fix #6904 logic errors in prefix graph data structure (#6929)

### Non-User-Facing Changes

- fix several lgtm.com flags (#6757, #6883)
- cleanups and refactors for conda 4.5 (#6889)
- unify location of record types in conda/models/records.py (#6924)
- resolve #6952 memoize url search in package cache loading (#6957)

## 4.4.98 4.4.11 (2018-02-23)

### Improvements

- resolve #6582 swallow_broken_pipe context manager and Spinner refactor (#6616)
- resolve #6882 document max_shlvl (#6892)
- resolve #6733 make empty env vars sequence-safe for sequence parameters (#6741)
- resolve #6900 don't record conda skeleton environments in environments.txt (#6908)

### Bug Fixes

- fix potential error in ensure_pad(); add more tests (#6817)
- fix #6840 handle error return values in conda.sh (#6850)
- use conda.gateways.disk for misc.py imports (#6870)
- fix #6672 don't update conda during conda-env operations (#6773)
- fix #6811 don't attempt copy/remove fallback for rename failures (#6867)
- fix #6667 aliased posix commands (#6669)
- fix #6816 fish environment autocomplete (#6885)
- fix #6880 build_number comparison not functional in match_spec (#6881)
- fix #6910 sort key prioritizes build string over build number (#6911)
- fix #6914, #6691 conda can fail to update packages even though newer versions exist (#6921)
- fix #6899 handle Unicode output in activate commands (#6909)

## 4.4.99 4.4.10 (2018-02-09)

### Bug Fixes

- fix #6837 require at least futures 3.0.0 (#6855)
- fix #6852 ensure temporary path is writable (#6856)
- fix #6833 improve feature mismatch metric (via 4.3.34 #6853)

### 4.4.100 4.4.9 (2018-02-06)

#### Improvements

- resolve #6632 display package removal plan when deleting an env (#6801)

#### Bug Fixes

- fix #6531 don't drop credentials for conda-build workaround (#6798)
- fix external command execution issue (#6789)
- fix #5792 conda env export error common in path (#6795)
- fix #6390 add CorruptedEnvironmentError (#6778)
- fix #5884 allow --insecure CLI flag without contradicting meaning of ssl_verify (#6782)
- fix MatchSpec.match() accepting dict (#6808)
- fix broken Anaconda Prompt for users with spaces in paths (#6825)
- JSONDecodeError was added in Python 3.5 (#6848)
- fix #6796 update PATH/prompt on reactivate (#6828)
- fix #6401 non-ascii characters on windows using expanduser (#6847)
- fix #6824 import installers before invoking any (#6849)

### 4.4.101 4.4.8 (2018-01-25)

#### Improvements

- allow falsey values for default_python to avoid pinning python (#6682)
- resolve #6700 add message for no space left on device (#6709)
- make variable 'sourced' local for posix shells (#6726)
- add column headers to conda list results (#5726)

#### Bug Fixes

- fix #6713 allow parenthesis in prefix path for conda.bat (#6722)
- fix #6684 --force message (#6723)
- fix #6693 KeyError with '–update-deps' (#6694)
- fix aggressive_update_packages availability (#6727)
- fix #6745 don't truncate channel priority map in conda installer (#6746)
- add workaround for system Python usage by lsb_release (#6769)
- fix #6624 can't start new thread (#6653)
- fix #6628 'conda install --rev' in conda 4.4 (#6724)
- fix #6707 FileNotFoundError when extracting tarball (#6708)

- fix #6704 unexpected token in conda.bat (#6710)
- fix #6208 return for no pip in environment (#6784)
- fix #6457 env var cleanup (#6790)
- fix #6645 escape paths for argparse help (#6779)
- fix #6739 handle unicode in environment variables for py2 activate (#6777)
- fix #6618 RepresenterError with 'conda config --set' (#6619)
- fix #6699 suppress memory error upload reports (#6776)
- fix #6770 CRLF for cmd.exe (#6775)
- fix #6514 add message for case-insensitive filesystem errors (#6764)
- fix #6537 AttributeError value for url not set (#6754)
- fix #6748 only warn if unable to register environment due to EACCES (#6752)

### 4.4.102 4.4.7 (2018-01-08)

#### Improvements

- resolve #6650 add upgrade message for unicode errors in python 2 (#6651)

#### Bug Fixes

- fix #6643 difference between '==' and 'exact_match_' (#6647)
- fix #6620 KeyError(u'CONDA_PREFIX',) (#6652)
- fix #6661 remove env from environments.txt (#6662)
- fix #6629 'conda update --name' AssertionError (#6656)
- fix #6630 repodata AssertionError (#6657)
- fix #6626 add setuptools as constrained dependency (#6654)
- fix #6659 conda list explicit should be dependency sorted (#6671)
- fix #6665 KeyError for channel '' (#6668, #6673)
- fix #6627 AttributeError on 'conda activate' (#6655)

### 4.4.103 4.4.6 (2017-12-31)

#### Bug Fixes

- fix #6612 do not assume Anaconda Python on Windows nor Library\bin hack (#6615)
- recipe test improvements and associated bug fixes (#6614)

### 4.4.104 4.4.5 (2017-12-29)

#### Bug Fixes

- fix #6577, #6580 single quote in PS1 (#6585)
- fix #6584 os.getcwd() FileNotFound (#6589)
- fix #6592 deactivate command order (#6602)
- fix #6579 python not recognized as command (#6588)
- fix #6572 cached repodata PermissionsError (#6573)
- change instances of 'root' to 'base' (#6598)
- fix #6607 use subprocess rather than execv for conda command extensions (#6609)
- fix #6581 git-bash activation (#6587)
- fix #6599 space in path to base prefix (#6608)

### 4.4.105 4.4.4 (2017-12-24)

#### Improvements

- add SUDO_ env vars to info reports (#6563)
- add additional information to the #6546 exception (#6551)

#### Bug Fixes

- fix #6548 'conda update' installs packages not in prefix #6550
- fix #6546 update after creating an empty env (#6568)
- fix #6557 conda list FileNotFoundError (#6558)
- fix #6554 package cache FileNotFoundError (#6555)
- fix #6529 yaml parse error (#6560)
- fix #6562 repodata_record.json permissions error stack trace (#6564)
- fix #6520 --use-local flag (#6526)

### 4.4.106 4.4.3 (2017-12-22)

#### Improvements

- adjust error report message (#6534)

### Bug Fixes

- fix #6530 package cache JsonDecodeError / ValueError (#6533)
- fix #6538 BrokenPipeError (#6540)
- fix #6532 remove anaconda metapackage hack (#6539)
- fix #6536 'conda env export' for old versions of pip (#6535)
- fix #6541 py2 and unicode in environments.txt (#6542)

### Non-User-Facing Changes

- regression tests for #6512 (#6515)

## 4.4.107 4.4.2 (2017-12-22)

### Deprecations/Breaking Changes

- resolve #6523 don't prune with --update-all (#6524)

### Bug Fixes

- fix #6508 environments.txt permissions error stack trace (#6511)
- fix #6522 error message formatted incorrectly (#6525)
- fix #6516 hold channels over from get_index to install_actions (#6517)

## 4.4.108 4.4.1 (2017-12-21)

### Bug Fixes

- fix #6512 reactivate does not accept arguments (#6513)

## 4.4.109 4.4.0 (2017-12-20)

### Recommended change to enable conda in your shell

With the release of conda 4.4, we recommend a change to how the `conda` command is made available to your shell environment. All the old methods still work as before, but you'll need the new method to enable the new `conda activate` and `conda deactivate` commands.

For the "Anaconda Prompt" on Windows, there is no change.

For Bourne shell derivatives (bash, zsh, dash, etc.), you likely currently have a line similar to

```
export PATH="/opt/conda/bin:$PATH"
```

in your `~/.bashrc` file (or `~/.bash_profile` file on macOS). The effect of this line is that your base environment is put on PATH, but without actually *activating* that environment. (In 4.4 we've renamed the 'root' environment to the 'base' environment.) With conda 4.4, we recommend removing the line where the `PATH` environment variable is modified, and replacing it with

```
. /opt/conda/etc/profile.d/conda.sh
conda activate base
```

In the above, it's assumed that `/opt/conda` is the location where you installed miniconda or Anaconda. It may also be something like `~/Anaconda3` or `~/miniconda2`.

For system-wide conda installs, to make the `conda` command available to all users, rather than manipulating individual `~/.bashrc` (or `~/.bash_profile`) files for each user, just execute once

```
$ sudo ln -s /opt/conda/etc/profile.d/conda.sh /etc/profile.d/conda.sh
```

This will make the `conda` command itself available to all users, but conda's base (root) environment will *not* be activated by default. Users will still need to run `conda activate base` to put the base environment on PATH and gain access to the executables in the base environment.

After updating to conda 4.4, we also recommend pinning conda to a specific channel. For example, executing the command

```
$ conda config --system --add pinned_packages conda-canary::conda
```

will make sure that whenever conda is installed or changed in an environment, the source of the package is always being pulled from the `conda-canary` channel. This will be useful for people who use `conda-forge`, to prevent conda from flipping back and forth between 4.3 and 4.4.

### New Feature Highlights

- **conda activate**: The logic and mechanisms underlying environment activation have been reworked. With conda 4.4, `conda activate` and `conda deactivate` are now the preferred commands for activating and deactivating environments. You'll find they are much more snappy than the `source activate` and `source deactivate` commands from previous conda versions. The `conda activate` command also has advantages of (1) being universal across all OSes, shells, and platforms, and (2) not having path collisions with scripts from other packages like python virtualenv's activate script.
- **constrained, optional dependencies**: Conda now allows a package to constrain versions of other packages installed alongside it, even if those constrained packages are not themselves hard dependencies for that package. In other words, it lets a package specify that, if another package ends up being installed into an environment, it must at least conform to a certain version specification. In effect, constrained dependencies are a type of "reverse" dependency. It gives a tool to a parent package to exclude other packages from an environment that might otherwise want to depend on it.

  Constrained optional dependencies are supported starting with conda-build 3.0 (via [conda/conda-build#2001[(https://github.com/conda/conda-build/pull/2001)). A new `run_constrained` keyword, which takes a list of package specs similar to the `run` keyword, is recognized under the `requirements` section of `meta.yaml`. For backward compatibility with versions of conda older than 4.4, a requirement may be listed in both the `run` and the `run_constrained` section. In that case older versions of conda will see the package as a hard dependency, while conda 4.4 will understand that the package is meant to be optional.

  Optional, constrained dependencies end up in `repodata.json` under a `constrains` keyword, parallel to the `depends` keyword for a package's hard dependencies.
- **enhanced package query language**: Conda has a built-in query language for searching for and matching packages, what we often refer to as `MatchSpec`. The MatchSpec is what users input on the command line when they specify packages for `create`, `install`, `update`, and `remove` operations. With this release, MatchSpec (rather than a regex) becomes the default input for `conda search`. We have also substantially enhanced our MatchSpec query language.

  For example,

```
conda install conda-forge::python
```

is now a valid command, which specifies that regardless of the active list of channel priorities, the python package itself should come from the `conda-forge` channel. As before, the difference between `python=3.5` and `python==3.5` is that the first contains a "*fuzzy*" version while the second contains an *exact* version. The fuzzy spec will match all python packages with versions `>=3.5` and `<3.6`. The exact spec will match only python packages with version `3.5`, `3.5.0`, `3.5.0.0`, etc. The canonical string form for a MatchSpec is thus

```
(channel::)name(version(build_string))
```

which should feel natural to experienced conda users. Specifications however are often necessarily more complicated than this simple form can support, and for these situations we've extended the specification to include an optional square bracket `[]` component containing comma-separated key-value pairs to allow matching on most any field contained in a package's metadata. Take, for example,

```
conda search 'conda-forge/linux-64::*[md5=e42a03f799131d5af4196ce31a1084a7]' --info
```

which results in information for the single package

```
cytoolz 0.8.2 py35_0
--------------------
file name   : cytoolz-0.8.2-py35_0.tar.bz2
name        : cytoolz
version     : 0.8.2
build string: py35_0
build number: 0
size        : 1.1 MB
arch        : x86_64
platform    : Platform.linux
license     : BSD 3-Clause
subdir      : linux-64
url         : https://conda.anaconda.org/conda-forge/linux-64/cytoolz-0.8.2-py35_0.
↪tar.bz2
md5         : e42a03f799131d5af4196ce31a1084a7
dependencies:
  - python 3.5*
  - toolz >=0.8.0
```

The square bracket notation can also be used for any field that we match on outside the package name, and will override information given in the "simple form" position. To give a contrived example, `python==3.5[version='>=2.7,<2.8']` will match `2.7.*` versions and not `3.5`.

- **environments track user-requested state**: Building on our enhanced MatchSpec query language, conda environments now also track and differentiate (a) packages added to an environment because of an explicit user request from (b) packages brought into an environment to satisfy dependencies. For example, executing

```
conda install conda-forge::scikit-learn
```

will confine all future changes to the scikit-learn package in the environment to the conda-forge channel, until the spec is changed again. A subsequent command `conda install scikit-learn=0.18` would drop the `conda-forge` channel restriction from the package. And in this case, scikit-learn is the only user-defined spec, so the solver chooses dependencies from all configured channels and all available versions.

- **errors posted to core maintainers**: In previous versions of conda, unexpected errors resulted in a request for users to consider posting the error as a new issue on conda's github issue tracker. In conda 4.4, we've implemented

a system for users to opt-in to sending that same error report via an HTTP POST request directly to the core maintainers.

When an unexpected error is encountered, users are prompted with the error report followed by a `[y/N]` input. Users can elect to send the report, with 'no' being the default response. Users can also permanently opt-in or opt-out, thereby skipping the prompt altogether, using the boolean `report_errors` configuration parameter.

- **various UI improvements**: To push through some of the big leaps with transactions in conda 4.3, we accepted some regressions on progress bars and other user interface features. All of those indicators of progress, and more, have been brought back and further improved.
- **aggressive updates**: Conda now supports an `aggressive_update_packages` configuration parameter that holds a sequence of MatchSpec strings, in addition to the `pinned_packages` configuration parameter. Currently, the default value contains the packages `ca-certificates`, `certifi`, and `openssl`. When manipulating configuration with the `conda config` command, use of the `--system` and `--env` flags will be especially helpful here. For example,

```
conda config --add aggressive_update_packages defaults::pyopenssl --system
```

would ensure that, system-wide, solves on all environments enforce using the latest version of `pyopenssl` from the `defaults` channel.

```
conda config --add pinned_packages python=2.7 --env
```

would lock all solves for the current active environment to python versions matching `2.7.*`.

- **other configuration improvements**: In addition to `conda config --describe`, which shows detailed descriptions and default values for all available configuration parameters, we have a new `conda config --write-default` command. This new command simply writes the contents of `conda config --describe` to a condarc file, which is a great starter template. Without additional arguments, the command will write to the `.condarc` file in the user's home directory. The command also works with the `--system`, `--env`, and `--file` flags to write the contents to alternate locations.

  Conda exposes a tremendous amount of flexibility via configuration. For more information, The Conda Configuration Engine for Power Users blog post is a good resource.

### Deprecations/Breaking Changes

- the conda 'root' environment is now generally referred to as the 'base' environment
- Conda 4.4 now warns when available information about per-path sha256 sums and file sizes do not match the recorded information. The warning is scheduled to be an error in conda 4.5. Behavior is configurable via the `safety_checks` configuration parameter.
- remove support for with_features_depends (#5191)
- resolve #5468 remove --alt-hint from CLI API (#5469)
- resolve #5834 change default value of 'allow_softlinks' from True to False (#5835)
- resolve #5842 add deprecation warnings for 'conda env upload' and 'conda env attach' (#5843)

### API

- Add Solver from conda.core.solver with three methods to conda.api (4.4.0rc1) (#5838)

### Improvements

- constrained, optional dependencies (#4982)
- conda shell function (#5044, #5141, #5162, #5169, #5182, #5210, #5482)
- resolve #5160 conda xontrib plugin (#5157)
- resolve #1543 add support and tests for --no-deps and --only-deps (#5265)
- resolve #988 allow channel name to be part of the package name spec (#5365, #5791)
- resolve #5530 add ability for users to choose to post unexpected errors to core maintainers (#5531, #5571, #5585)
- Solver, UI, History, and Other (#5546, #5583, #5740)
- improve 'conda search' to leverage new MatchSpec query language (#5597)
- filter out unwritable package caches from conda clean command (#4620)
- envs_manager, requested spec history, declarative solve, and private env tests (#4676, #5114, #5094, #5145, #5492)
- make python entry point format match pip entry points (#5010)
- resolve #5113 clean up CLI imports to improve process startup time (#4799)
- resolve #5121 add features/track_features support for MatchSpec (#5054)
- resolve #4671 hold verify backoff count in transaction context (#5122)
- resolve #5078 record package metadata after tarball extraction (#5148)
- resolve #3580 support stacking environments (#5159)
- resolve #3763, #4378 allow pip requirements.txt syntax in environment files (#3969)
- resolve #5147 add 'config files' to conda info (#5269)
- use --format=json to parse list of pip packages (#5205)
- resolve #1427 remove startswith '.' environment name constraint (#5284)
- link packages from extracted tarballs when tarball is gone (#5289)
- resolve #2511 accept config information from stdin (#5309)
- resolve #4302 add ability to set map parameters with conda config (#5310)
- resolve #5256 enable conda config --get for all primitive parameters (#5312)
- resolve #1992 add short flag -C for --use-index-cache (#5314)
- resolve #2173 add --quiet option to conda clean (#5313)
- resolve #5358 conda should exec to subcommands, not subprocess (#5359)
- resolve #5411 add 'conda config --write-default' (#5412)
- resolve #5081 make pinned packages optional dependencies (#5414)
- resolve #5430 eliminate current deprecation warnings (#5422)
- resolve #5470 make stdout/stderr capture in python_api customizable (#5471)

- logging simplifications/improvements (#5547, #5578)
- update license information (#5568)
- enable threadpool use for repodata collection by default (#5546, #5587)
- conda info now raises PackagesNotFoundError (#5655)
- index building optimizations (#5776)
- fix #5811 change safety_checks default to 'warn' for conda 4.4 (4.4.0rc1) (#5824)
- add constrained dependencies to conda's own recipe (4.4.0rc1) (#5823)
- clean up parser imports (4.4.0rc2) (#5844)
- resolve #5983 add --download-only flag to create, install, and update (4.4.0rc2) (#5988)
- add ca-certificates and certifi to aggressive_update_packages default (4.4.0rc2) (#5994)
- use environments.txt to list all known environments (4.4.0rc2) (#6313)
- resolve #5417 ensure unlink order is correctly sorted (4.4.0) (#6364)
- resolve #5370 index is only prefix and cache in --offline mode (4.4.0) (#6371)
- reduce redundant sys call during file copying (4.4.0rc3) (#6421)
- enable aggressive_update_packages (4.4.0rc3) (#6392)
- default conda.sh to dash if otherwise can't detect (4.4.0rc3) (#6414)
- canonicalize package names when comparing with pip (4.4.0rc3) (#6438)
- add target prefix override configuration parameter (4.4.0rc3) (#6413)
- resolve #6194 warn when conda is outdated (4.4.0rc3) (#6370)
- add information to displayed error report (4.4.0rc3) (#6437)
- csh wrapper (4.4.0) (#6463)
- resolve #5158 --override-channels (4.4.0) (#6467)
- fish update for conda 4.4 (4.4.0) (#6475, #6502)
- skip an unnecessary environments.txt rewrite (4.4.0) (#6495)

### Bug Fixes

- fix some conda-build compatibility issues (#5089)
- resolve #5123 export toposort (#5124)
- fix #5132 signal handler can only be used in main thread (#5133)
- fix orphaned --clobber parser arg (#5188)
- fix #3814 don't remove directory that's not a conda environment (#5204)
- fix #4468 _license stack trace (#5206)
- fix #4987 conda update --all no longer displays full list of packages (#5228)
- fix #3489 don't error on remove --all if environment doesn't exist (#5231)
- fix #1509 bash doesn't need full path for pre/post link/unlink scripts on unix (#5252)
- fix #462 add regression test (#5286)

- fix #5288 confirmation prompt doesn't accept no (#5291)
- fix #1713 'conda package -w' is case dependent on Windows (#5308)
- fix #5371 try falling back to pip's vendored requests if no requests available (#5372)
- fix #5356 skip root logger configuration (#5380)
- fix #5466 scrambled URL of non-alias channel with token (#5467)
- fix #5444 environment.yml file not found (#5475)
- fix #3200 use proper unbound checks in bash code and test (#5476)
- invalidate PrefixData cache on rm_rf for conda-build (#5491, #5499)
- fix exception when generating JSON output (#5628)
- fix target prefix determination (#5642)
- use proxy to avoid segfaults (#5716)
- fix #5790 incorrect activation message (4.4.0rc1) (#5820)
- fix #5808 assertion error when loading package cache (4.4.0rc1) (#5815)
- fix #5809 _pip_install_via_requirements got an unexpected keyword argument 'prune' (4.4.0rc1) (#5814)
- fix #5811 change safety_checks default to 'warn' for conda 4.4 (4.4.0rc1) (#5824)
- fix #5825 --json output format (4.4.0rc1) (#5831)
- fix force_reinstall for case when packages aren't actually installed (4.4.0rc1) (#5836)
- fix #5680 empty pip subsection error in environment.yml (4.4.0rc2) (#6275)
- fix #5852 bad tokens from history crash conda installs (4.4.0rc2) (#6076)
- fix #5827 no error message on invalid command (4.4.0rc2) (#6352)
- fix exception handler for 'conda activate' (4.4.0rc2) (#6365)
- fix #6173 double prompt immediately after conda 4.4 upgrade (4.4.0rc2) (#6351)
- fix #6181 keep existing pythons pinned to minor version (4.4.0rc2) (#6363)
- fix #6201 incorrect subdir shown for conda search when package not found (4.4.0rc2) (#6367)
- fix #6045 help message and zsh shift (4.4.0rc3) (#6368)
- fix noarch python package resintall (4.4.0rc3) (#6394)
- fix #6366 shell activation message (4.4.0rc3) (#6369)
- fix #6429 AttributeError on 'conda remove' (4.4.0rc3) (#6434)
- fix #6449 problems with 'conda info --envs' (#6451)
- add debug exception for #6430 (4.4.0rc3) (#6435)
- fix #6441 NotImplementedError on 'conda list' (4.4.0rc3) (#6442)
- fix #6445 scale back directory activation in PWD (4.4.0rc3) (#6447)
- fix #6283 no-deps for conda update case (4.4.0rc3) (#6448)
- fix #6419 set PS1 in python code (4.4.0rc3) (#6446)
- fix #6466 sp_dir doesn't exist (#6470)
- fix #6350 --update-all removes too many packages (4.4.0) (#6491)

- fix #6057 unlink-link order for python noarch packages on windows 4.4.x (4.4.0) (#6494)

### Non-User-Facing Changes

- eliminate index modification in Resolve init (#4333)
- new MatchSpec implementation (#4158, #5517)
- update conda.recipe for 4.4 (#5086)
- resolve #5118 organization and cleanup for 4.4 release (#5115)
- remove unused disk space check instructions (#5167)
- localfs adapter tests (#5181)
- extra config command tests (#5185)
- add coverage for confirm (#5203)
- clean up FileNotFoundError and DirectoryNotFoundError (#5237)
- add assertion that a path only has a single hard link before rewriting prefixes (#5305)
- remove pycrypto as requirement on windows (#5326)
- import cleanup, dead code removal, coverage improvements, and other housekeeping (#5472, #5474, #5480)
- rename CondaFileNotFoundError to PathNotFoundError (#5521)
- work toward repodata API (#5267)
- rename PackageNotFoundError to PackagesNotFoundError and fix message formatting (#5602)
- update conda 4.4 bld.bat windows recipe (#5573)
- remove last remnant of CondaEnvRuntimeError (#5643)
- fix typo (4.4.0rc2) (#6043)
- replace Travis-CI with CircleCI (4.4.0rc2) (#6345)
- key-value features (#5645); reverted in 4.4.0rc2 (#6347, #6492)
- resolve #6431 always add env_vars to info_dict (4.4.0rc3) (#6436)
- move shell inside conda directory (4.4.0) (#6479)
- remove dead code (4.4.0) (#6489)

## 4.4.110  4.3.34 (2018-02-09)

### Bug Fixes

- fix #6833 improve feature mismatch metric (#6853)

### 4.4.111 4.3.33 (2018-01-24)

#### Bug Fixes

- fix #6718 broken 'conda install --rev' (#6719)
- fix #6765 adjust the feature score assigned to packages not installed (#6766)

### 4.4.112 4.3.32 (2018-01-10)

#### Improvements

- resolve #6711 fall back to copy/unlink for EINVAL, EXDEV rename failures (#6712)

#### Bug Fixes

- fix #6057 unlink-link order for python noarch packages on windows (#6277)
- fix #6509 custom_channels incorrect in 'conda config --show' (#6510)

### 4.4.113 4.3.31 (2017-12-15)

#### Improvements

- add delete_trash to conda_env create (#6299)

#### Bug Fixes

- fix #6023 assertion error for temp file (#6154)
- fix #6220 --no-builds flag for 'conda env export' (#6221)
- fix #6271 timestamp prioritization results in undesirable race-condition (#6279)

#### Non-User-Facing Changes

- fix two failing integration tests after anaconda.org API change (#6182)
- resolve #6243 mark root as not writable when sys.prefix is not a conda environment (#6274)
- add timing instrumentation (#6458)

### 4.4.114 4.3.30 (2017-10-17)

#### Improvements

- address #6056 add additional proxy variables to 'conda info --all' (#6083)

### Bug Fixes

- address #6164 move add_defaults_to_specs after augment_specs (#6172)
- fix #6057 add additional detail for message 'cannot link source that does not exist' (#6082)
- fix #6084 setting default_channels from CLI raises NotImplementedError (#6085)

## 4.4.115 4.3.29 (2017-10-09)

### Bug Fixes

- fix #6096 coerce to millisecond timestamps (#6131)

## 4.4.116 4.3.28 (2017-10-06)

### Bug Fixes

- fix #5854 remove imports of pkg_resources (#5991)
- fix millisecond timestamps (#6001)

## 4.4.117 4.3.27 (2017-09-18)

### Bug Fixes

- fix #5980 always delete_prefix_from_linked_data in rm_rf (#5982)

## 4.4.118 4.3.26 (2017-09-15)

### Deprecations/Breaking Changes

- resolve #5922 prioritize channels within multi-channels (#5923)
- add https://repo.continuum.io/pkgs/main to defaults multi-channel (#5931)

### Improvements

- add a channel priority minimization pass to solver logic (#5859)
- invoke cmd.exe with /D for pre/post link/unlink scripts (#5926)
- add boto3 use to s3 adapter (#5949)

### Bug Fixes

- always remove linked prefix entry with rm_rf (#5846)
- resolve #5920 bump repodata pickle version (#5921)
- fix msys2 activate and deactivate (#5950)

## 4.4.119 4.3.25 (2017-08-16)

### Deprecations/Breaking Changes

- resolve #5834 change default value of 'allow_softlinks' from True to False (#5839)

### Improvements

- add non-admin check to optionally disable non-privileged operation (#5724)
- add extra warning message to always_softlink configuration option (#5826)

### Bug Fixes

- fix #5763 channel url string splitting error (#5764)
- fix regex for repodata _mod and _etag (#5795)
- fix uncaught OSError for missing device (#5830)

## 4.4.120 4.3.24 (2017-07-31)

### Bug Fixes

- fix #5708 package priority sort order (#5733)

## 4.4.121 4.3.23 (2017-07-21)

### Improvements

- resolve #5391 PackageNotFound and NoPackagesFoundError clean up (#5506)

### Bug Fixes

- fix #5525 too many Nones in CondaHttpError (#5526)
- fix #5508 assertion failure after test file not cleaned up (#5533)
- fix #5523 catch OSError when home directory doesn't exist (#5549)
- fix #5574 traceback formatting (#5580)
- fix #5554 logger configuration levels (#5555)
- fix #5649 create_default_packages configuration (#5703)

### 4.4.122 4.3.22 (2017-06-12)

#### Improvements

- resolve #5428 clean up cli import in conda 4.3.x (#5429)
- resolve #5302 add warning when creating environment with space in path (#5477)
- for ftp connections, ignore host IP from PASV as it is often wrong (#5489)
- expose common race condition exceptions in exports for conda-build (#5498)

#### Bug Fixes

- fix #5451 conda clean --json bug (#5452)
- fix #5400 confusing deactivate message (#5473)
- fix #5459 custom subdir channel parsing (#5478)
- fix #5483 problem with setuptools / pkg_resources import (#5496)

### 4.4.123 4.3.21 (2017-05-25)

#### Bug Fixes

- fix #5420 conda-env update error (#5421)
- fix #5425 is admin on win int not callable (#5426)

### 4.4.124 4.3.20 (2017-05-23)

#### Improvements

- resolve #5217 skip user confirm in python_api, force always_yes (#5404)

#### Bug Fixes

- fix #5367 conda info always shows 'unknown' for admin indicator on Windows (#5368)
- fix #5248 drop plan description information that might not always be accurate (#5373)
- fix #5378 duplicate log messages (#5379)
- fix #5298 record has 'build', not 'build_string' (#5382)
- fix #5384 silence logging info to avoid interfering with JSON output (#5393)
- fix #5356 skip root/conda logger init for cli.python_api (#5405)

### Non-User-Facing Changes

- avoid persistent state after channel priority test (#5392)
- resolve #5402 add regression test for #5384 (#5403)
- clean up inner function definition inside for loop (#5406)

## 4.4.125 4.3.19 (2017-05-18)

### Improvements

- resolve #3689 better error messaging for missing anaconda-client (#5276)
- resolve #4795 conda env export lacks -p flag (#5275)
- resolve #5315 add alias verify_ssl for ssl_verify (#5316)
- resolve #3399 add netrc existence/location to 'conda info' (#5333)
- resolve #3810 add --prefix to conda env update (#5335)

### Bug Fixes

- fix #5272 conda env export ugliness under python2 (#5273)
- fix #4596 warning message from pip on conda env export (#5274)
- fix #4986 --yes not functioning for conda clean (#5311)
- fix #5329 unicode errors on Windows (#5328, #5357)
- fix sys_prefix_unfollowed for Python 3 (#5334)
- fix #5341 --json flag with conda-env (#5342)
- fix 5321 ensure variable PROMPT is set in activate.bat (#5351)

### Non-User-Facing Changes

- test conda 4.3 with requests 2.14.2 (#5281)
- remove pycrypto as requirement on windows (#5325)
- fix typo avaialble -> available (#5345)
- fix test failures related to menuinst update (#5344, #5362)

## 4.4.126 4.3.18 (2017-05-09)

### Improvements

- resolve #4224 warn when pysocks isn't installed (#5226)
- resolve #5229 add --insecure flag to skip ssl verification (#5230)
- resolve #4151 add admin indicator to conda info on windows (#5241)

### Bug Fixes

- fix #5152 conda info spacing (#5166)
- fix --use-index-cache actually hitting the index cache (#5134)
- backport LinkPathAction verify from 4.4 (#5171)
- fix #5184 stack trace on invalid map configuration parameter (#5186)
- fix #5189 stack trace on invalid sequence config param (#5192)
- add support for the linux-aarch64 platform (#5190)
- fix repodata fetch with the `--offline` flag (#5146)
- fix #1773 conda remove spell checking (#5176)
- fix #3470 reduce excessive error messages (#5195)
- fix #1597 make extra sure --dry-run doesn't take any actions (#5201)
- fix #3470 extra newlines around exceptions (#5200)
- fix #5214 install messages for 'nothing_to_do' case (#5216)
- fix #598 stack trace for condarc write permission denied (#5232)
- fix #4960 extra information when exception can't be displayed (#5236)
- fix #4974 no matching dist in linked data for prefix (#5239)
- fix #5258 give correct element types for conda config --describe (#5259)
- fix #4911 separate shutil.copy2 into copy and copystat (#5261)

### Non-User-Facing Changes

- resolve #5138 add test of rm_rf of symlinked files (#4373)
- resolve #4516 add extra trace-level logging (#5249, #5250)
- add tests for --update-deps flag (#5264)

## 4.4.127  4.3.17 (2017-04-24)

### Improvements

- fall back to copy if hardlink fails (#5002)
- add timestamp metadata for tiebreaking conda-build 3 hashed packages (#5018)
- resolve #5034 add subdirs configuration parameter (#5030)
- resolve #5081 make pinned packages optional/constrained dependencies (#5088)
- resolve #5108 improve behavior and add tests for spaces in paths (#4786)

### Bug Fixes

- quote prefix paths for locations with spaces (#5009)
- remove binstar logger configuration overrides (#4989)
- fix #4969 error in DirectoryNotFoundError (#4990)
- fix #4998 pinned string format (#5011)
- fix #5039 collecting main_info shouldn't fail on requests import (#5090)
- fix #5055 improve bad token message for anaconda.org (#5091)
- fix #5033 only re-register valid signal handlers (#5092)
- fix #5028 imports in main_list (#5093)
- fix #5073 allow client_ssl_cert{_key} to be of type None (#5096)
- fix #4671 backoff for package validate race condition (#5098)
- fix #5022 gnu_get_libc_version => linux_get_libc_version (#5099)
- fix #4849 package name match bug (#5103)
- fixes #5102 allow proxy_servers to be of type None (#5107)
- fix #5111 incorrect typify for str + NoneType (#5112)

### Non-User-Facing Changes

- resolve #5012 remove CondaRuntimeError and RuntimeError (#4818)
- full audit ensuring relative import paths within project (#5090)
- resolve #5116 refactor conda/cli/activate.py to help menuinst (#4406)

## 4.4.128 4.3.16 (2017-03-30)

### Improvements

- additions to configuration SEARCH_PATH to improve consistency (#4966)
- add 'conda config --describe' and extra config documentation (#4913)
- enable packaging pinning in condarc using pinned_packages config parameter as beta feature (#4921, #4964)

### Bug Fixes

- fix #4914 handle directory creation on top of file paths (#4922)
- fix #3982 issue with CONDA_ENV and using powerline (#4925)
- fix #2611 update instructions on how to source conda.fish (#4924)
- fix #4860 missing information on package not found error (#4935)
- fix #4944 command not found error error (#4963)

### 4.4.129 4.3.15 (2017-03-20)

#### Improvements

- allow pkgs_dirs to be configured using `conda config` (#4895)

#### Bug Fixes

- remove incorrect elision of delete_prefix_from_linked_data() (#4814)
- fix envs_dirs order for read-only root prefix (#4821)
- fix break-point in conda clean (#4801)
- fix long shebangs when creating entry points (#4828)
- fix spelling and typos (#4868, #4869)
- fix #4840 TypeError reduce() of empty sequence with no initial value (#4843)
- fix zos subdir (#4875)
- fix exceptions triggered during activate (#4873)

### 4.4.130 4.3.14 (2017-03-03)

#### Improvements

- use cPickle in place of pickle for repodata (#4717)
- ignore pyc compile failure (#4719)
- use conda.exe for windows entry point executable (#4716, #4720)
- localize use of conda_signal_handler (#4730)
- add skip_safety_checks configuration parameter (#4767)
- never symlink executables using ORIGIN (#4625)
- set activate.bat codepage to CP_ACP (#4558)

#### Bug Fixes

- fix #4777 package cache initialization speed (#4778)
- fix #4703 menuinst PathNotFoundException (#4709)
- ignore permissions error if user_site can't be read (#4710)
- fix #4694 don't import requests directly in models (#4711)
- fix #4715 include resources directory in recipe (#4716)
- fix CondaHttpError for URLs that contain '%' (#4769)
- bug fixes for preferred envs (#4678)
- fix #4745 check for info/index.json with package is_extracted (#4776)
- make sure url gets included in CondaHTTPError (#4779)

- fix #4757 map-type configs set to None (#4774)
- fix #4788 partial package extraction (#4789)

#### Non-User-Facing Changes

- test coverage improvement (#4607)
- CI configuration improvements (#4713, #4773, #4775)
- allow sha256 to be None (#4759)
- add cache_fn_url to exports (#4729)
- add unicode paths for PY3 integration tests (#4760)
- additional unit tests (#4728, #4783)
- fix conda-build compatibility and tests (#4785)

### 4.4.131 4.3.13 (2017-02-17)

#### Improvements

- resolve #4636 environment variable expansion for pkgs_dirs (#4637)
- link, symlink, islink, and readlink for Windows (#4652, #4661)
- add extra information to CondaHTTPError (#4638, #4672)

#### Bug Fixes

- maximize requested builds after feature determination (#4647)
- fix #4649 incorrect assert statement concerning package cache directory (#4651)
- multi-user mode bug fixes (#4663)

#### Non-User-Facing Changes

- path_actions unit tests (#4654)
- remove dead code (#4369, #4655, #4660)
- separate repodata logic from index into a new core/repodata.py module (#4669)

### 4.4.132 4.3.12 (2017-02-14)

#### Improvements

- prepare conda for uploading to pypi (#4619)
- better general http error message (#4627)
- disable old python noarch warning (#4576)

### Bug Fixes

- fix UnicodeDecodeError for ensure_text_type (#4585)
- fix determination of if file path is writable (#4604)
- fix #4592 BufferError cannot close exported pointers exist (#4628)
- fix run_script current working directory (#4629)
- fix pkgs_dirs permissions regression (#4626)

### Non-User-Facing Changes

- fixes for tests when conda-bld directory doesn't exist (#4606)
- use requirements.txt and Makefile for travis-ci setup (#4600, #4633)
- remove hasattr use from compat functions (#4634)

## 4.4.133 4.3.11 (2017-02-09)

### Bug Fixes

- fix attribute error in add_defaults_to_specs (#4577)

## 4.4.134 4.3.10 (2017-02-07)

### Improvements

- remove .json from pickle path (#4498)
- improve empty repodata noarch warning and error messages (#4499)
- don't add python and lua as default specs for private envs (#4529, #4533)
- let default_python be None (#4547, #4550)

### Bug Fixes

- fix #4513 null pointer exception for channel without noarch (#4518)
- fix ssl_verify set type (#4517)
- fix bug for windows multiuser (#4524)
- fix clone with noarch python packages (#4535)
- fix ipv6 for python 2.7 on Windows (#4554)

### Non-User-Facing Changes

- separate integration tests with a marker (#4532)

## 4.4.135 4.3.9 (2017-01-31)

### Improvements

- improve repodata caching for performance (#4478, #4488)
- expand scope of packages included by bad_installed (#4402)
- silence pre-link warning for old noarch (#4451)
- add configuration to optionally require noarch repodata (#4450)
- improve conda subprocessing (#4447)
- respect info/link.json (#4482)

### Bug Fixes

- fix #4398 'hard' was used for link type at one point (#4409)
- fixed "No matches for wildcard '$activate_d/*.fish'" warning (#4415)
- print correct activate/deactivate message for fish shell (#4423)
- fix 'Dist' object has no attribute 'fn' (#4424)
- fix noarch generic and add additional integration test (#4431)
- fix #4425 unknown encoding (#4433)

### Non-User-Facing Changes

- fail CI on conda-build fail (#4405)
- run doctests (#4414)
- make index record mutable again (#4461)
- additional test for conda list --json (#4480)

## 4.4.136 4.3.8 (2017-01-23)

### Bug Fixes

- fix #4309 ignore EXDEV error for directory renames (#4392)
- fix #4393 by force-renaming certain backup files if the path already exists (#4397)

### 4.4.137 4.3.7 (2017-01-20)

#### Bug Fixes

- actually revert json output for leaky plan (#4383)
- fix not raising on pre/post-link error (#4382)
- fix find_commands and find_executable for symlinks (#4387)

### 4.4.138 4.3.6 (2017-01-18)

#### Bug Fixes

- fix 'Uncaught backoff with errno 41' warning on windows (#4366)
- revert json output for leaky plan (#4349)
- audit os.environ setting (#4360)
- fix #4324 using old dist string instead of dist object (#4361)
- fix #4351 infinite recursion via code in #4120 (#4370)
- fix #4368 conda -h (#4367)
- workaround for symlink race conditions on activate (#4346)

### 4.4.139 4.3.5 (2017-01-17)

#### Improvements

- add exception message for corrupt repodata (#4315)

#### Bug Fixes

- fix package not being found in cache after download (#4297)
- fix logic for Content-Length mismatch (#4311, #4326)
- use unicode_escape after etag regex instead of utf-8 (#4325)
- fix #4323 central condarc file being ignored (#4327)
- fix #4316 a bug in deactivate (#4316)
- pass target_prefix as env_prefix regardless of is_unlink (#4332)
- pass positional argument 'context' to BasicClobberError (#4335)

### Non-User-Facing Changes

- additional package pinning tests (#4317)

## 4.4.140 4.3.4 (2017-01-13)

### Improvements

- vendor url parsing from urllib3 (#4289)

### Bug Fixes

- fix some bugs in windows multi-user support (#4277)
- fix problems with channels of type (#4290)
- include aliases for first command-line argument (#4279)
- fix for multi-line FTP status codes (#4276)

### Non-User-Facing Changes

- make arch in IndexRecord a StringField instead of EnumField
- improve conda-build compatibility (#4266)

## 4.4.141 4.3.3 (2017-01-10)

### Improvements

- respect Cache-Control max-age header for repodata (#4220)
- add 'local_repodata_ttl' configurability (#4240)
- remove questionable "nothing to install" logic (#4237)
- relax channel noarch requirement for 4.3; warn now, raise in future feature release (#4238)
- add additional info to setup.py warning message (#4258)

### Bug Fixes

- remove features properly (#4236)
- do not use `IFS` to find activate/deactivate scripts to source (#4239)
- fix #4235 print message to stderr (#4241)
- fix relative path to python in activate.bat (#4242)
- fix args.channel references (#4245, #4246)
- ensure cache_fn_url right pad (#4255)
- fix #4256 subprocess calls must have env wrapped in str (#4259)

### 4.4.142 4.3.2 (2017-01-06)

#### Deprecations/Breaking Changes

- Further refine conda channels specification. To verify if the url of a channel represents a valid conda channel, we check that `noarch/repodata.json` and/or `noarch/repodata.json.bz2` exist, even if empty. (#3739)

#### Improvements

- add new 'path_conflict' and 'clobber' configuration options (#4119)
- separate fetch/extract pass for explicit URLs (#4125)
- update conda homepage to conda.io (#4180)

#### Bug Fixes

- fix pre/post unlink/link scripts (#4113)
- fix package version regex and bug in create_link (#4132)
- fix history tracking (#4143)
- fix index creation order (#4131)
- fix #4152 conda env export failure (#4175)
- fix #3779 channel UNC path encoding errors on windows (#4190)
- fix progress bar (#4191)
- use context.channels instead of args.channel (#4199)
- don't use local cached repodata for file:// urls (#4209)

#### Non-User-Facing Changes

- xfail anaconda token test if local token is found (#4124)
- fix open-ended test failures relating to python 3.6 release (#4145)
- extend timebomb for test_multi_channel_export (#4169)
- don't unlink dists that aren't in the index (#4130)
- add python 3.6 and new conda-build test targets (#4194)

### 4.4.143 4.3.1 (2016-12-19)

#### Improvements

- additional pre-transaction validation (#4090)
- export FileMode enum for conda-build (#4080)
- memoize disk permissions tests (#4091)
- local caching of repodata without remote server calls; new 'repodata_timeout_secs' configuration parameter (#4094)

- performance tuning (#4104)
- add additional fields to dist object serialization (#4102)

### Bug Fixes

- fix a noarch install bug on windows (#4071)
- fix a spec mismatch that resulted in python versions getting mixed during packaging (#4079)
- fix rollback linked record (#4092)
- fix #4097 keep split in PREFIX_PLACEHOLDER (#4100)

## 4.4.144 4.3.0 (2016-12-14) Safety

### New Features

- **Unlink and Link Packages in a Single Transaction**: In the past, conda hasn't always been safe and defensive with its disk-mutating actions. It has gleefully clobbered existing files, and mid-operation failures leave environments completely broken. In some of the most severe examples, conda can appear to "uninstall itself." With this release, the unlinking and linking of packages for an executed command is done in a single transaction. If a failure occurs for any reason while conda is mutating files on disk, the environment will be returned its previous state. While we've implemented some pre-transaction checks (verifying package integrity for example), it's impossible to anticipate every failure mechanism. In some circumstances, OS file permissions cannot be fully known until an operation is attempted and fails. And conda itself is not without bugs. Moving forward, unforeseeable failures won't be catastrophic. (#3833, #4030)
- **Progressive Fetch and Extract Transactions**: Like package unlinking and linking, the download and extract phases of package handling have also been given transaction-like behavior. The distinction is the rollback on error is limited to a single package. Rather than rolling back the download and extract operation for all packages, the single-package rollback prevents the need for having to re-download every package if an error is encountered. (#4021, #4030)
- **Generic- and Python-Type Noarch/Universal Packages**: Along with conda-build 2.1.0, a noarch/universal type for python packages is officially supported. These are much like universal python wheels. Files in a python noarch package are linked into a prefix just like any other conda package, with the following additional features

  1. conda maps the `site-packages` directory to the correct location for the python version in the environment,
  2. conda maps the python-scripts directory to either $PREFIX/bin$or PREFIX/Scripts depending on platform,
  3. conda creates the python entry points specified in the conda-build recipe, and
  4. conda compiles pyc files at install time when prefix write permissions are guaranteed.

  Python noarch packages must be "fully universal." They cannot have OS- or python version-specific dependencies. They cannot have OS- or python version-specific "scripts" files. If these features are needed, traditional conda packages must be used. (#3712)
- **Multi-User Package Caches**: While the on-disk package cache structure has been preserved, the core logic implementing package cache handling has had a complete overhaul. Writable and read-only package caches are fully supported. (#4021)
- **Python API Module**: An oft requested feature is the ability to use conda as a python library, obviating the need to "shell out" to another python process. Conda 4.3 includes a `conda.cli.python_api` module that facilitates this use case. While we maintain the user-facing command-line interface, conda commands can be executed in-process. There is also a `conda.exports` module to facilitate longer-term usage of conda as a library across

conda conda releases. However, conda's python code *is* considered internal and private, subject to change at any time across releases. At the moment, conda will not install itself into environments other than its original install environment. (#4028)

- **Remove All Locks**: Locking has never been fully effective in conda, and it often created a false sense of security. In this release, multi-user package cache support has been implemented for improved safety by hard-linking packages in read-only caches to the user's primary user package cache. Still, users are cautioned that undefined behavior can result when conda is running in multiple process and operating on the same package caches and/or environments. (#3862)

### Deprecations/Breaking Changes

- Conda will refuse to clobber existing files that are not within the unlink instructions of the transaction. At the risk of being user-hostile, it's a step forward for conda. We do anticipate some growing pains. For example, conda will not clobber packages that have been installed with pip (or any other package manager). In other instances, conda packages that contain overlapping file paths but are from different package families will not install at the same time. The `--force` command line flag is the escape hatch. Using `--force` will let your operation proceed, but also makes clear that you want conda to do something it considers unsafe.
- Conda signed packages have been removed in 4.3. Vulnerabilities existed. An illusion of security is worse than not having the feature at all. We will be incorporating The Update Framework into conda in a future feature release. (#4064)
- Conda 4.4 will drop support for older versions of conda-build.

### Improvements

- create a new "trace" log level enabled by `-v -v -v` or `-vvv` (#3833)
- allow conda to be installed with pip, but only when used as a library/dependency (#4028)
- the 'r' channel is now part of defaults (#3677)
- private environment support for conda (#3988)
- support v1 info/paths.json file (#3927, #3943)
- support v1 info/package_metadata.json (#4030)
- improved solver hint detection, simplified filtering (#3597)
- cache VersionOrder objects to improve performance (#3596)
- fix documentation and typos (#3526, #3572, #3627)
- add multikey configuration validation (#3432)
- some Fish autocompletions (#2519)
- reduce priority for packages removed from the index (#3703)
- add user-agent, uid, gid to conda info (#3671)
- add conda.exports module (#3429)
- make http timeouts configurable (#3832)
- add a pkgs_dirs config parameter (#3691)
- add an 'always_softlink' option (#3870, #3876)
- pre-checks for diskspace, etc for fetch and extract #(4007)

- address #3879 don't print activate message when quiet config is enabled (#3886)
- add zos-z subdir (#4060)
- add elapsed time to HTTP errors (#3942)

### Bug Fixes

- account for the Windows Python 2.7 os.environ unicode aversion (#3363)
- fix link field in record object (#3424)
- anaconda api token bug fix; additional tests (#3673)
- fix #3667 unicode literals and unicode decode (#3682)
- add conda-env entrypoint (#3743)
- fix #3807 json dump on conda config --show --json (#3811)
- fix #3801 location of temporary hard links of index.json (#3813)
- fix invalid yml example (#3849)
- add arm platforms back to subdirs (#3852)
- fix #3771 better error message for assertion errors (#3802)
- fix #3999 spaces in shebang replacement (#4008)
- config --show-sources shouldn't show force by default (#3891)
- fix #3881 don't install conda-env in clones of root (#3899)
- conda-build dist compatibility (#3909)

### Non-User-Facing Changes

- remove unnecessary eval (#3428)
- remove dead install_tar function (#3641)
- apply PEP-8 to conda-env (#3653)
- refactor dist into an object (#3616)
- vendor appdirs; remove conda's dependency on anaconda-client import (#3675)
- revert boto patch from #2380 (#3676)
- move and update ROOT_NO_RM (#3697)
- integration tests for conda clean (#3695, #3699)
- disable coverage on s3 and ftp requests adapters (#3696, #3701)
- github repo hygiene (#3705, #3706)
- major install refactor (#3712)
- remove test timebombs (#4012)
- LinkType refactor (#3882)
- move CrossPlatformStLink and make available as export (#3887)
- make Record immutable (#3965)

- project housekeeping (#3994, #4065)
- context-dependent setup.py files (#4057)

### 4.4.145 4.2.17 (unreleased)

#### Improvements

- silence pre-link warning for old noarch 4.2.x backport (#4453)

#### Bug Fixes

- remove incorrect elision of delete_prefix_from_linked_data() (#4813)
- fix CB #1825 context clobbering (#4867)
- fix #5101 api->conda regex substitution for Anaconda API channels (#5100)

#### Non-User-Facing Changes

- build 4.2.x against conda-build 2.1.2 and enforce passing (#4462)

### 4.4.146 4.2.16 (2017-01-20)

#### Improvements

- vendor url parsing from urllib3 (#4289)
- workaround for symlink race conditions on activate (#4346)

#### Bug Fixes

- do not replace \ with / in file:// URLs on Windows (#4269)
- include aliases for first command-line argument (#4279)
- fix for multi-line FTP status codes (#4276)
- fix errors with unknown type channels (#4291)
- change sys.exit to raise UpgradeError when info/files not found (#4388)

#### Non-User-Facing Changes

- start using doctests in test runs and coverage (#4304)
- additional package pinning tests (#4312)

### 4.4.147 4.2.15 (2017-01-10)

#### Improvements

- use 'post' instead of 'dev' for commits according to PEP-440 (#4234)
- do not use IFS to find activate/deactivate scripts to source (#4243)
- fix relative path to python in activate.bat (#4244)

#### Bug Fixes

- replace sed with python for activate and deactivate #4257

### 4.4.148 4.2.14 (2017-01-07)

#### Improvements

- use install.rm_rf for TemporaryDirectory cleanup (#3425)
- improve handling of local dependency information (#2107)
- add default channels to exports for Windows and Unix (#4103)
- make subdir configurable (#4178)

#### Bug Fixes

- fix conda/install.py single-file behavior (#3854)
- fix the api->conda substitution (#3456)
- fix silent directory removal (#3730)
- fix location of temporary hard links of index.json (#3975)
- fix potential errors in multi-channel export and offline clone (#3995)
- fix auxlib/packaging, git hashes are not limited to 7 characters (#4189)
- fix compatibility with requests >=2.12, add pyopenssl as dependency (#4059)
- fix #3287 activate in 4.1-4.2.3 clobbers non-conda PATH changes (#4211)

#### Non-User-Facing Changes

- fix open-ended test failures relating to python 3.6 release (#4166)
- allow args passed to cli.main() (#4193, #4200, #4201)
- test against python 3.6 (#4197)

### 4.4.149 4.2.13 (2016-11-22)

#### Deprecations/Breaking Changes

- show warning message for pre-link scripts (#3727)
- error and exit for install of packages that require conda minimum version 4.3 (#3726)

#### Improvements

- double/extend http timeouts (#3831)
- let descriptive http errors cover more http exceptions (#3834)
- backport some conda-build configuration (#3875)

#### Bug Fixes

- fix conda/install.py single-file behavior (#3854)
- fix the api->conda substitution (#3456)
- fix silent directory removal (#3730)
- fix #3910 null check for is_url (#3931)

#### Non-User-Facing Changes

- flake8 E116, E121, & E123 enabled (#3883)

### 4.4.150 4.2.12 (2016-11-02)

#### Bug Fixes

- fix #3732, #3471, #3744 CONDA_BLD_PATH (#3747)
- fix #3717 allow no-name channels (#3748)
- fix #3738 move conda-env to ruamel_yaml (#3740)
- fix conda-env entry point (#3745 via #3743)
- fix again #3664 trash emptying (#3746)

### 4.4.151 4.2.11 (2016-10-23)

#### Improvements

- only try once for windows trash removal (#3698)

### Bug Fixes

- fix anaconda api token bug (#3674)
- fix #3646 FileMode enum comparison (#3683)
- fix #3517 conda install --mkdir (#3684)
- fix #3560 hack anaconda token coverup on conda info (#3686)
- fix #3469 alias envs_path to envs_dirs (#3685)

## 4.4.152 4.2.10 (2016-10-18)

### Improvements

- add json output for `conda info -s` (#3588)
- ignore certain binary prefixes on windows (#3539)
- allow conda config files to have .yaml extensions or 'condarc' anywhere in filename (#3633)

### Bug Fixes

- fix conda-build's handle_proxy_407 import (#3666)
- fix #3442, #3459, #3481, #3531, #3548 multiple networking and auth issues (#3550)
- add back linux-ppc64le subdir support (#3584)
- fix #3600 ensure links are removed when unlinking (#3625)
- fix #3602 search channels by platform (#3629)
- fix duplicated packages when updating environment (#3563)
- fix #3590 exception when parsing invalid yaml (#3593 via #3634)
- fix #3655 a string decoding error (#3656)

### Non-User-Facing Changes

- backport conda.exports module to 4.2.x (#3654)
- travis-ci OSX fix (#3615 via #3657)

## 4.4.153 4.2.9 (2016-09-27)

### Bug Fixes

- fix #3536 conda-env messaging to stdout with --json flag (#3537)
- fix #3525 writing to sys.stdout with --json flag for post-link scripts (#3538)
- fix #3492 make NULL falsey with python 3 (#3524)

### 4.4.154 4.2.8 (2016-09-26)

#### Improvements

- add “error” key back to json error output (#3523)

#### Bug Fixes

- fix #3453 conda fails with create_default_packages (#3454)
- fix #3455 --dry-run fails (#3457)
- dial down error messages for rm_rf (#3522)
- fix #3467 AttributeError encountered for map config parameter validation (#3521)

### 4.4.155 4.2.7 (2016-09-16)

#### Deprecations/Breaking Changes

- revert to 4.1.x behavior of `conda list --export` (#3450, #3451)

#### Bug Fixes

- don’t add binstar token if it’s given in the channel spec (#3427, #3440, #3444)
- fix #3433 failure to remove broken symlinks (#3436)

#### Non-User-Facing Changes

- use install.rm_rf for TemporaryDirectory cleanup (#3425)

### 4.4.156 4.2.6 (2016-09-14)

#### Improvements

- add support for client TLS certificates (#3419)
- address #3267 allow migration of channel_alias (#3410)
- conda-env version matches conda version (#3422)

#### Bug Fixes

- fix #3409 unsatisfiable dependency error message (#3412)
- fix #3408 quiet rm_rf (#3413)
- fix #3407 padding error messaging (#3416)
- account for the Windows Python 2.7 os.environ unicode aversion (#3363 via #3420)

### 4.4.157 4.2.5 (2016-09-08)

#### Deprecations/Breaking Changes

- partially revert #3041 giving conda config --add previous --prepend behavior (#3364 via #3370)
- partially revert #2760 adding back conda package command (#3398)

#### Improvements

- order output of conda config --show; make --json friendly (#3384 via #3386)
- clean the pid based lock on exception (#3325)
- improve file removal on all platforms (#3280 via #3396)

#### Bug Fixes

- fix #3332 allow download urls with :: in them (#3335)
- fix always_yes and not-set argparse args overriding other sources (#3374)
- fix ftp fetch timeout (#3392)
- fix #3307 add try/except block for touch lock (#3326)
- fix CONDA_CHANNELS environment variable splitting (#3390)
- fix #3378 CONDA_FORCE_32BIT environment variable (#3391)
- make conda info channel urls actually give urls (#3397)
- fix cio_test compatibility (#3395 via #3400)

### 4.4.158 4.2.4 (2016-08-18)

#### Bug Fixes

- fix #3277 conda list package order (#3278)
- fix channel priority issue with duplicated channels (#3283)
- fix local channel channels; add full conda-build unit tests (#3281)
- fix conda install with no package specified (#3284)
- fix #3253 exporting and importing conda environments (#3286)
- fix priority messaging on conda config --get (#3304)
- fix conda list --export; additional integration tests (#3291)
- fix conda update --all idempotence; add integration tests for channel priority (#3306)

### Non-User-Facing Changes

- additional conda-env integration tests (#3288)

## 4.4.159 4.2.3 (2016-08-11)

### Improvements

- added zsh and zsh.exe to Windows shells (#3257)

### Bug Fixes

- allow conda to downgrade itself (#3273)
- fix breaking changes to conda-build from 4.2.2 (#3265)
- fix empty environment issues with conda and conda-env (#3269)

### Non-User-Facing Changes

- add integration tests for conda-env (#3270)
- add more conda-build smoke tests (#3274)

## 4.4.160 4.2.2 (2016-08-09)

### Improvements

- enable binary prefix replacement on windows (#3262)
- add `--verbose` command line flag (#3237)
- improve logging and exception detail (#3237, #3252)
- do not remove empty environment without asking; raise an error when a named environment can't be found (#3222)

### Bug Fixes

- fix #3226 user condarc not available on Windows (#3228)
- fix some bugs in conda config --show* (#3212)
- fix conda-build local channel bug (#3202)
- remove subprocess exiting message (#3245)
- fix comment parsing and channels in conda-env environment.yml (#3258, #3259)
- fix context error with conda-env (#3232)
- fix #3182 conda install silently skipping failed linking (#3184)

### 4.4.161 4.2.1 (2016-08-01)

#### Improvements

- improve an error message that can happen during conda install --revision (#3181)
- use clean sys.exit with user choice 'No' (#3196)

#### Bug Fixes

- critical fix for 4.2.0 error when no git is on PATH (#3193)
- revert #3171 lock cleaning on exit pending further refinement
- patches for conda-build compatibility with 4.2 (#3187)
- fix a bug in --show-sources output that ignored aliased parameter names (#3189)

#### Non-User-Facing Changes

- move scripts in bin to shell directory (#3186)

### 4.4.162 4.2.0 (2016-07-28) Configuration

#### New Features

- **New Configuration Engine**: Configuration and "operating context" are the foundation of conda's functionality. Conda now has the ability to pull configuration information from a multitude of on-disk locations, including `.d` directories and a `.condarc` file *within* a conda environment), along with full `CONDA_` environment variable support. Helpful validation errors are given for improperly-specified configuration. Full documentation updates pending. (#2537, #3160, #3178)
- **New Exception Handling Engine**: Previous releases followed a pattern of premature exiting (with hard calls to `sys.exit()` when exceptional circumstances were encountered. This release replaces over 100 `sys.exit` calls with python exceptions. For conda developers, this will result in tests that are easier to write. For developers using conda, this is a first step on a long path toward conda being directly importable. For conda users, this will eventually result in more helpful and descriptive errors messages. (#2899, #2993, #3016, #3152, #3045)
- **Empty Environments**: Conda can now create "empty" environments when no initial packages are specified, alleviating a common source of confusion. (#3072, #3174)
- **Conda in Private Env**: Conda can now be configured to live within its own private environment. While it's not yet default behavior, this represents a first step toward separating the `root` environment into a "conda private" environment and a "user default" environment. (#3068)
- **Regex Version Specification**: Regular expressions are now valid version specifiers. For example, `^1\.[5-8]\.1$|2.2`. (#2933)

### Deprecations/Breaking Changes

- remove conda init (#2759)
- remove conda package and conda bundle (#2760)
- deprecate conda-env repo; pull into conda proper (#2950, #2952, #2954, #3157, #3163, #3170)
- force use of ruamel_yaml (#2762)
- implement conda config --prepend; change behavior of --add to --append (#3041)
- exit on link error instead of logging it (#2639)

### Improvements

- improve locking (#2962, #2989, #3048, #3075)
- clean up requests usage for fetching packages (#2755)
- remove excess output from conda --help (#2872)
- remove os.remove in update_prefix (#3006)
- better error behavior if conda is spec'd for a non-root environment (#2956)
- scale back try_write function on unix (#3076)

### Bug Fixes

- remove psutil requirement, fixes annoying error message (#3135, #3183)
- fix #3124 add threading lock to memoize (#3134)
- fix a failure with multi-threaded repodata downloads (#3078)
- fix windows file url (#3139)
- address #2800, error with environment.yml and non-default channels (#3164)

### Non-User-Facing Changes

- project structure enhancement (#2929, #3132, #3133, #3136)
- clean up channel handling with new channel model (#3130, #3151)
- add Anaconda Cloud / Binstar auth handler (#3142)
- remove dead code (#2761, #2969)
- code refactoring and additional tests (#3052, #3020)
- remove auxlib from project root (#2931)
- vendor auxlib 0.0.40 (#2932, #2943, #3131)
- vendor toolz 0.8.0 (#2994)
- move progressbar to vendor directory (#2951)
- fix conda.recipe for new quirks with conda-build (#2959)
- move captured function to common module (#3083)

- rename CHANGELOG to md (#3087)

### 4.4.163 4.1.13 (unreleased)

- improve handling of local dependency information, #2107
- show warning message for pre-link scripts, #3727
- error and exit for install of packages that require conda minimum version 4.3, #3726
- fix conda/install.py single-file behavior, #3854
- fix open-ended test failures relating to python 3.6 release, #4167
- fix #3287 activate in 4.1-4.2.3 clobbers non-conda PATH changes, #4211
- fix relative path to python in activate.bat, #4244

### 4.4.164 4.1.12 (2016-09-08)

- fix #2837 "File exists" in symlinked path with parallel activations, #3210
- fix prune option when installing packages, #3354
- change check for placeholder to be more friendly to long PATH, #3349

### 4.4.165 4.1.11 (2016-07-26)

- fix PS1 backup in activate script, #3135 via #3155
- correct resolution for 'handle failures in binstar_client more generally', #3156

### 4.4.166 4.1.10 (2016-07-25)

- ignore symlink failure because of read-only file system, #3055
- backport shortcut tests, #3064
- fix #2979 redefinition of $SHELL variable, #3081
- fix #3060 --clone root --copy exception, #3080

### 4.4.167 4.1.9 (2016-07-20)

- fix #3104, add global BINSTAR_TOKEN_PAT
- handle failures in binstar_client more generally

### 4.4.168 4.1.8 (2016-07-12)

- fix #3004 UNAUTHORIZED for url (null binstar token), #3008
- fix overwrite existing redirect shortcuts when symlinking envs, #3025
- partially revert no default shortcuts, #3032, #3047

### 4.4.169 4.0.11 2016-07-09

- allow auto_update_conda from sysrc, #3015 via #3021

### 4.4.170 4.1.7 (2016-07-07)

- add msys2 channel to defaults on Windows, #2999
- fix #2939 channel_alias issues; improve offline enforcement, #2964
- fix #2970, #2974 improve handling of file:// URLs inside channel, #2976

### 4.4.171 4.1.6 (2016-07-01)

- slow down exp backoff from 1 ms to 100 ms factor, #2944
- set max time on exp_backoff to ~6.5 sec,#2955
- fix #2914 add/subtract from PATH; kill folder output text, #2917
- normalize use of get_index behavior across clone/explicit, #2937
- wrap root prefix check with normcase, #2938

### 4.4.172 4.1.5 (2016-06-29)

- more conservative auto updates of conda #2900
- fix some permissions errors with more aggressive use of move_path_to_trash, #2882
- fix #2891 error if allow_other_channels setting is used, #2896
- fix #2886, #2907 installing a tarball directly from the package cache, #2908
- fix #2681, #2778 reverting #2320 lock behavior changes, #2915

### 4.4.173 4.0.10 (2016-06-29)

- fix #2846 revert the use of UNC paths; shorten trash filenames, #2859 via #2878
- fix some permissions errors with more aggressive use of move_path_to_trash, #2882 via #2894

### 4.4.174 4.1.4 (2016-06-27)

- fix #2846 revert the use of UNC paths; shorten trash filenames, #2859
- fix exp backoff on Windows, #2860
- fix #2845 URL for local file repos, #2862
- fix #2764 restore full path var on win; create to CONDA_PREFIX env var, #2848
- fix #2754 improve listing pip installed packages, #2873
- change root prefix detection to avoid clobbering root activate scripts, #2880
- address #2841 add lowest and highest priority indication to channel config output, #2875
- add SYMLINK_CONDA to planned instructions, #2861
- use CONDA_PREFIX, not CONDA_DEFAULT_ENV for activate.d, #2856
- call scripts with redirect on win; more error checking to activate, #2852

### 4.4.175 4.1.3 (2016-06-23)

- ensure conda-env auto update, along with conda, #2772
- make yaml booleans behave how everyone expects them to, #2784
- use accept-encoding for repodata; prefer repodata.json to repodata.json.bz2, #2821
- additional integration and regression tests, #2757, #2774, #2787
- add offline mode to printed info; use offline flag when grabbing channels, #2813
- show conda-env version in conda info, #2819
- adjust channel priority superseded list, #2820
- support epoch ! characters in command line specs, #2832
- accept old default names and new ones when canonicalizing channel URLs #2839
- push PATH, PS1 manipulation into shell scripts, #2796
- fix #2765 broken source activate without arguments, #2806
- fix standalone execution of install.py, #2756
- fix #2810 activating conda environment broken with git bash on Windows, #2795
- fix #2805, #2781 handle both file-based channels and explicit file-based URLs, #2812
- fix #2746 conda create --clone of root, #2838
- fix #2668, #2699 shell recursion with activate #2831

### 4.4.176 4.1.2 (2016-06-17)

- improve messaging for “downgrades” due to channel priority, #2718
- support conda config channel append/prepend, handle duplicates, #2730
- remove --shortcuts option to internal CLI code, #2723
- fix an issue concerning space characters in paths in activate.bat, #2740
- fix #2732 restore yes/no/on/off for booleans on the command line, #2734
- fix #2642 tarball install on Windows, #2729
- fix #2687, #2697 WindowsError when creating environments on Windows, #2717
- fix #2710 link instruction in conda create causes TypeError, #2715
- revert #2514, #2695, disabling of .netrc files, #2736
- revert #2281 printing progress bar to terminal, #2707

### 4.4.177 4.1.1 (2016-06-16)

- add auto_update_conda config parameter, #2686
- fix #2669 conda config --add channels can leave out defaults, #2670
- fix #2703 ignore activate symlink error if links already exist, #2705
- fix #2693 install duplicate packages with older version of Anaconda, #2701
- fix #2677 respect HTTP_PROXY, #2695
- fix #2680 broken fish integration, #2685, #2694
- fix an issue with conda never exiting, #2689
- fix #2688 explicit file installs, #2708
- fix #2700 conda list UnicodeDecodeError, #2706

### 4.4.178 4.0.9 (2016-06-15)

- add auto_update_conda config parameter, #2686

### 4.4.179 4.1.0 (2016-06-14) Channel Priority

- clean up activate and deactivate scripts, moving back to conda repo, #1727, #2265, #2291, #2473, #2501, #2484
- replace pyyaml with ruamel_yaml, #2283, #2321
- better handling of channel collisions, #2323, #2369 #2402, #2428
- improve listing of pip packages with conda list, #2275
- re-license progressbar under BSD 3-clause, #2334
- reduce the amount of extraneous info in hints, #2261
- add --shortcuts option to install shortcuts on windows, #2623
- skip binary replacement on windows, #2630

- don't show channel urls by default in conda list, #2282
- package resolution and solver tweaks, #2443, #2475, #2480
- improved version & build matching, #2442, #2488
- print progress to the terminal rather than stdout, #2281
- verify version specs given on command line are valid, #2246
- fix for try_write function in case of odd permissions, #2301
- fix a conda search --spec error, #2343
- update User-Agent for conda connections, #2347
- remove some dead code paths, #2338, #2374
- fixes a thread safety issue with http requests, #2377, #2383
- manage BeeGFS hard-links non-POSIX configuration, #2355
- prevent version downgrades during removes, #2394
- fix conda info --json, #2445
- truncate shebangs over 127 characters using /usr/bin/env, #2479
- extract packages to a temporary directory then rename, #2425, #2483
- fix help in install, #2460
- fix re-install bug when sha1 differs, #2507
- fix a bug with file deletion, #2499
- disable .netrc files, #2514
- dont fetch index on remove --all, #2553
- allow track_features to be a string *or* a list in .condarc, #2541
- fix #2415 infinite recursion in invalid_chains, #2566
- allow channel_alias to be different than binstar, #2564

### 4.4.180 4.0.8 (2016-06-03)

- fix a potential problem with moving files to trash, #2587

### 4.4.181 4.0.7 (2016-05-26)

- workaround for boto bug, #2380

### 4.4.182 4.0.6 (2016-05-11)

- log "custom" versions as updates rather than downgrades, #2290
- fixes a TypeError exception that can occur on install/update, #2331
- fixes an error on Windows removing files with long path names, #2452

### 4.4.183 4.0.5 (2016-03-16)

- improved help documentation for install, update, and remove, #2262
- fixes #2229 and #2250 related to conda update errors on Windows, #2251
- fixes #2258 conda list for pip packages on Windows, #2264

### 4.4.184 4.0.4 (2016-03-10)

- revert #2217 closing request sessions, #2233

### 4.4.185 4.0.3 (2016-03-10)

- adds a `conda clean --all` feature, #2211
- solver performance improvements, #2209
- fixes conda list for pip packages on windows, #2216
- quiets some logging for package downloads under python 3, #2217
- more urls for `conda list --explicit`, #1855
- prefer more "latest builds" for more packages, #2227
- fixes a bug with dependency resolution and features, #2226

### 4.4.186 4.0.2 (2016-03-08)

- fixes track_features in ~/.condarc being a list, see also #2203
- fixes incorrect path in lock file error #2195
- fixes issues with cloning environments, #2193, #2194
- fixes a strange interaction between features and versions, #2206
- fixes a bug in low-level SAT clause generation creating a preference for older versions, #2199

### 4.4.187 4.0.1 (2016-03-07)

- fixes an install issue caused by md5 checksum mismatches, #2183
- remove auxlib build dependency, #2188

### 4.4.188 4.0.0 (2016-03-04) Solver

- The solver has been retooled significantly. Performance should be improved in most circumstances, and a number of issues involving feature conflicts should be resolved.
- `conda update <package>` now handles dependencies properly according to the setting of the "update_deps" configuration: –update-deps: conda will also update any dependencies as needed to install the latest version of the requested packages. The minimal set of changes required to achieve this is sought. –no-update-deps: conda will update the packages *only* to the extent that no updates to the dependencies are required The previous behavior, which would update the packages without regard to their dependencies, could result in a broken configuration, and has been removed.
- Conda finally has an official logo.
- Fix `conda clean --packages` on Windows, #1944
- Conda sub-commands now support dashes in names, #1840

### 4.4.189 3.19.4 (unreleased)

- improve handling of local dependency information, #2107
- use install.rm_rf for TemporaryDirectory cleanup, #3425
- fix the api->conda substitution, #3456
- error and exit for install of packages that require conda minimum version 4.3, #3726
- show warning message for pre-link scripts, #3727
- fix silent directory removal, #3730
- fix conda/install.py single-file behavior, #3854

### 4.4.190 3.19.3 (2016-02-19)

- fix critical issue, see #2106

### 4.4.191 3.19.2 (2016-02-19)

- add basic activate/deactivate, conda activate/deactivate/ls for fish, see #545
- remove error when CONDA_FORCE_32BIT is set on 32-bit systems, #1985
- suppress help text for --unknown option, #2051
- fix issue with conda create --clone post-link scripts, #2007
- fix a permissions issue on windows, #2083

### 4.4.192 3.19.1 (2016-02-01)

- resolve.py: properly escape periods in version numbers, #1926
- support for pinning Lua by default, #1934
- remove hard-coded test URLs, a module cio_test is now expected when CIO_TEST is set

### 4.4.193 3.19.0 (2015-12-17)

- OpenBSD 5.x support, #1891
- improve install CLI to make Miniconda -f work, #1905

### 4.4.194 3.18.9 (2015-12-10)

- allow chaining default_channels (only applies to "system" condarc), from from CLI, #1886
- improve default for --show-channel-urls in conda list, #1900

### 4.4.195 3.18.8 (2015-12-03)

- always attempt to delete files in rm_rf, #1864

### 4.4.196 3.18.7 (2015-12-02)

- simplify call to menuinst.install()
- add menuinst as dependency on Windows
- add ROOT_PREFIX to post-link (and pre_unlink) environment

### 4.4.197 3.18.6 (2015-11-19)

- improve conda clean when user lacks permissions, #1807
- make show_channel_urls default to True, #1771
- cleaner write tests, #1735
- fix documentation, #1709
- improve conda clean when directories don't exist, #1808

### 4.4.198 3.18.5 (2015-11-11)

- fix bad menuinst exception handling, #1798
- add workaround for unresolved dependencies on Windows

### 4.4.199 3.18.4 (2015-11-09)

- allow explicit file to contain MD5 hashsums
- add --md5 option to "conda list --explicit"
- stop infinite recursion during certain resolve operations, #1749
- add dependencies even if strictness == 3, #1766

### 4.4.200 3.18.3 (2015-10-15)

- added a pruning step for more efficient solves, #1702
- disallow conda-env to be installed into non-root environment
- improve error output for bad command input, #1706
- pass env name and setup cmd to menuinst, #1699

### 4.4.201 3.18.2 (2015-10-12)

- add "conda list --explicit" which contains the URLs of all conda packages to be installed, and can used with the install/create --file option, #1688
- fix a potential issue in conda clean
- avoid issues with LookupErrors when updating Python in the root environment on Windows
- don't fetch the index from the network with conda remove
- when installing conda packages directly, "conda install .tar.bz2", unlink any installed package with that name (not just the installed one)
- allow menu items to be installed in non-root env, #1692

### 4.4.202 3.18.1 (2015-09-28)

- fix: removed reference to win_ignore_root in plan module

### 4.4.203 3.18.0 (2015-09-28)

- allow Python to be updated in root environment on Windows, #1657
- add defaults to specs after getting pinned specs (allows to pin a different version of Python than what is installed)
- show what older versions are in the solutions in the resolve debug log
- fix some issues with Python 3.5
- respect --no-deps when installing from .tar or .tar.bz2
- avoid infinite recursion with NoPackagesFound and conda update --all --file
- fix conda update --file
- toposort: Added special case to remove 'pip' dependency from 'python'
- show dotlog messages during hint generation with --debug

- disable the max_only heuristic during hint generation
- new version comparison algorithm, which consistently compares any version string, and better handles version strings using things like alpha, beta, rc, post, and dev. This should remove any inconsistent version comparison that would lead to conda installing an incorrect version.
- use the trash in rm_rf, meaning more things will get the benefit of the trash system on Windows
- add the ability to pass the --file argument multiple times
- add conda upgrade alias for conda update
- add update_dependencies condarc option and --update-deps/–no-update-deps command line flags
- allow specs with conda update --all
- add --show-channel-urls and --no-show-channel-urls command line options
- add always_copy condarc option
- conda clean properly handles multiple envs directories. This breaks backwards compatibility with some of the --json output. Some of the old –json keys are kept for backwards compatibility.

### 4.4.204 3.17.0 (2015-09-11)

- add windows_forward_slashes option to walk_prefix(), see #1513
- add ability to set CONDA_FORCE_32BIT environment variable, it should should only be used when running conda-build, #1555
- add config option to makes the python dependency on pip optional, #1577
- fix an UnboundLocalError
- print note about pinned specs in no packages found error
- allow wildcards in AND-connected version specs
- print pinned specs to the debug log
- fix conda create --clone with create_default_packages
- give a better error when a proxy isn't found for a given scheme
- enable running 'conda run' in offline mode
- fix issue where hardlinked cache contents were being overwritten
- correctly skip packages whose dependencies can't be found with conda update --all
- use clearer terminology in -m help text.
- use splitlines to break up multiple lines throughout the codebase
- fix AttributeError with SSLError

### 4.4.205 3.16.0 (2015-08-10)

- rename binstar -> anaconda, see #1458
- fix --use-local when the conda-bld directory doesn't exist
- fixed --offline option when using "conda create --clone", see #1487
- don't mask recursion depth errors
- add conda search --reverse-dependency
- check whether hardlinking is available before linking when using "python install.py --link" directly, see #1490
- don't exit nonzero when installing a package with no dependencies
- check which features are installed in an environment via track_features, not features
- set the verify flag directly on CondaSession (fixes conda skeleton not respecting the ssl_verify option)

### 4.4.206 3.15.1 (2015-07-23)

- fix conda with older versions of argcomplete
- restore the --force-pscheck option as a no-op for backwards compatibility

### 4.4.207 3.15.0 (2015-07-22)

- sort the output of conda info package correctly
- enable tab completion of conda command extensions using argcomplete. Command extensions that import conda should use conda.cli.conda_argparse.ArgumentParser instead of argparse.ArgumentParser. Otherwise, they should enable argcomplete completion manually.
- allow psutil and pycosat to be updated in the root environment on Windows
- remove all mentions of pscheck. The --force-pscheck flag has been removed.
- added support for S3 channels
- fix color issues from pip in conda list on Windows
- add support for other machine types on Linux, in particular ppc64le
- add non_x86_linux_machines set to config module
- allow ssl_verify to accept strings in addition to boolean values in condarc
- enable --set to work with both boolean and string values

### 4.4.208 3.14.1 (2015-06-29)

- make use of Crypto.Signature.PKCS1_PSS module, see #1388
- note when features are being used in the unsatisfiable hint

### 4.4.209 3.14.0 (2015-06-16)

- add ability to verify signed packages, see #1343 (and conda-build #430)
- fix issue when trying to add 'pip' dependency to old python packages
- provide option "conda info --unsafe-channels" for getting unobscured channel list, #1374

### 4.4.210 3.13.0 (2015-06-04)

- avoid the Windows file lock by moving files to a trash directory, #1133
- handle env dirs not existing in the Environments completer
- rename binstar.org -> anaconda.org, see #1348
- speed up 'source activate' by ~40%

### 4.4.211 3.12.0 (2015-05-05)

- correctly allow conda to update itself
- print which file leads to the "unable to remove file" error on Windows
- add support for the no_proxy environment variable, #1171
- add a much faster hint generation for unsatisfiable packages, which is now always enabled (previously it would not run if there were more than ten specs). The new hint only gives one set of conflicting packages, rather than all sets, so multiple passes may be necessary to fix such issues
- conda extensions that import conda should use conda.cli.conda_argparser.ArgumentParser instead of argparse.ArgumentParser to conform to the conda help guidelines (e.g., all help messages should be capitalized with periods, and the options should be preceded by "Options:" for the sake of help2man).
- add confirmation dialog to conda remove. Fixes conda remove --dry-run.

### 4.4.212 3.11.0 (2015-04-22)

- fix issue where forced update on Windows could cause a package to break
- remove detection of running processes that might conflict
- deprecate --force-pscheck (now a no-op argument)
- make conda search --outdated --names-only work, fixes #1252
- handle the history file not having read or write permissions better
- make multiple package resolutions warning easier to read
- add --full-name to conda list
- improvements to command help

### 4.4.213 3.10.1 (2015-04-06)

- fix logic in @memoized for unhashable args
- restored json cache of repodata, see #1249
- hide binstar tokens in conda info --json
- handle CIO_TEST='2 '
- always find the solution with minimal number of packages, even if there are many solutions
- allow comments at the end of the line in requirement files
- don't update the progressbar until after the item is finished running
- add conda/ to HTTP header User-Agent string

### 4.4.214 3.10.0 (2015-03-12)

- change default repo urls to be https
- add --offline to conda search
- add --names-only and --full-name to conda search
- add tab completion for packages to conda search

### 4.4.215 3.9.1 (2015-02-24)

- pscheck: check for processes in the current environment, see #1157
- don't write to the history file if nothing has changed, see #1148
- conda update --all installs packages without version restrictions (except for Python), see #1138
- conda update --all ignores the anaconda metapackage, see #1138
- use forward slashes for file urls on Windows
- don't symlink conda in the root environment from activate
- use the correct package name in the progress bar info
- use json progress bars for unsatisfiable dependencies hints
- don't let requests decode gz files when downloaded

### 4.4.216 3.9.0 (2015-02-16)

- remove (de)activation scripts from conda, those are now in conda-env
- pip is now always added as a Python dependency
- allow conda to be installed into environments which start with _
- add argcomplete tab completion for environments with the -n flag, and for package names with install, update, create, and remove

### 4.4.217 3.8.4 (2015-02-03)

- copy (de)activate scripts from conda-env
- Add noarch (sub) directory support

### 4.4.218 3.8.3 (2015-01-28)

- simplified how ROOT_PREFIX is obtained in (de)activate

### 4.4.219 3.8.2 (2015-01-27)

- add conda clean --source-cache to clean the conda build source caches
- add missing quotes in (de)activate.bat, fixes problem in Windows when conda is installed into a directory with spaces
- fix conda install --copy

### 4.4.220 3.8.1 (2015-01-23)

- add missing utf-8 decoding, fixes Python 3 bug when icondata to json file

### 4.4.221 3.8.0 (2015-01-22)

- move active script into conda-env, which is now a new dependency
- load the channel urls in the correct order when using concurrent.futures
- add optional 'icondata' key to json files in conda-meta directory, which contain the base64 encoded png file or the icon
- remove a debug print statement

### 4.4.222 3.7.4 (2014-12-18)

- add --offline option to install, create, update and remove commands, and also add ability to set "offline: True" in condarc file
- add conda uninstall as alias for conda remove
- add conda info --root
- add conda.pip module
- fix CONDARC pointing to non-existing file, closes issue #961
- make update -f work if the package is already up-to-date
- fix possible TypeError when printing an error message
- link packages in topologically sorted order (so that pre-link scripts can assume that the dependencies are installed)
- add --copy flag to install
- prevent the progressbar from crashing conda when fetching in some situations

### 4.4.223 3.7.3 (2014-11-05)

- conda install from a local conda package (or a tar fill which contains conda packages), will now also install the dependencies listed by the installed packages.
- add SOURCE_DIR environment variable in pre-link subprocess
- record all created environments in ~/.conda/environments.txt

### 4.4.224 3.7.2 (2014-10-31)

- only show the binstar install message once
- print the fetching repodata dot after the repodata is fetched
- write the install and remove specs to the history file
- add '-y' as an alias to '–yes'
- the `--file` option to conda config now defaults to os.environ.get('CONDARC')
- some improvements to documentation (–help output)
- add user_rc_path and sys_rc_path to conda info --json
- cache the proxy username and password
- avoid warning about conda in pscheck
- make ~/.conda/envs the first user envs dir

### 4.4.225 3.7.1 (2014-10-07)

- improve error message for forgetting to use source with activate and deactivate, see issue #601
- don't allow to remove the current environment, see issue #639
- don't fail if binstar_client can't be imported for other reasons, see issue #925
- allow spaces to be contained in conda run
- only show the conda install binstar hint if binstar is not installed
- conda info package_spec now gives detailed info on packages. conda info path has been removed, as it is duplicated by conda package -w path.

### 4.4.226 3.7.0 (2014-09-19)

- faster algorithm for --alt-hint
- don't allow channel_alias with allow_other_channels: false if it is set in the system .condarc
- don't show long "no packages found" error with update --all
- automatically add the Binstar token to urls when the binstar client is installed and logged in
- carefully avoid showing the binstar token or writing it to a file
- be more careful in conda config about keys that are the wrong type
- don't expect directories starting with conda- to be commands

- no longer recommend to run conda init after pip installing conda. A pip installed conda will now work without being initialized to create and manage other environments
- the rm function on Windows now works around access denied errors
- fix channel urls now showing with conda list with show_channel_urls set to true

### 4.4.227 3.6.4 (2014-09-08)

- fix removing packages that aren't in the channels any more
- Pretties output for --alt-hint

### 4.4.228 3.6.3 (2014-09-04)

- skip packages that can't be found with update --all
- add --use-local to search and remove
- allow --use-local to be used along with -c (–channels) and –override-channels. --override-channels now requires either -c or –use-local
- allow paths in has_prefix to be quoted, to allow for spaces in paths on Windows
- retain Unix style path separators for prefixes in has_prefix on Windows (if the placeholder path uses /, replace it with a path that uses /, not )
- fix bug in --use-local due to API changes in conda-build
- include user site directories in conda info -s
- make binary has_prefix replacement work with spaces after the prefix
- make binary has_prefix replacement replace multiple occurrences of the placeholder in the same null-terminated string
- don't show packages from other platforms as installed or cached in conda search
- be more careful about not warning about conda itself in pscheck
- Use a progress bar for the unsatisfiable packages hint generation
- Don't use TemporaryFile in try_write, as it is too slow when it fails
- Ignore InsecureRequestWarning when ssl_verify is False
- conda remove removes features tracked by removed packages in track_features

### 4.4.229 3.6.2 (2014-08-20)

- add --use-index-cache to conda remove
- fix a bug where features (like mkl) would be selected incorrectly
- use concurrent.future.ThreadPool to fetch package metadata asynchronously in Python 3.
- do the retries in rm_rf on every platform
- use a higher cutoff for package name misspellings
- allow changing default channels in "system" .condarc

### 4.4.230 3.6.1 (2014-08-13)

- add retries to download in fetch module
- improved error messages for missing packages
- more robust rm_rf on Windows
- print multiline help for subcommands correctly

### 4.4.231 3.6.0 (2014-08-11)

- correctly check if a package can be hard-linked if it isn't extracted yet
- change how the package plan is printed to better show what is new, updated, and downgraded
- use suggest_normalized_version in the resolve module. Now versions like 1.0alpha that are not directly recognized by verlib's NormalizedVersion are supported better
- conda run command, to run apps and commands from packages
- more complete --json API. Every conda command should fully support --json output now.
- show the conda_build and requests versions in conda info
- include packages from setup.py develop in conda list (with use_pip)
- raise a warning instead of dying when the history file is invalid
- use urllib.quote on the proxy password
- make conda search --outdated --canonical work
- pin the Python version during conda init
- fix some metadata that is written for Python during conda init
- allow comments in a pinned file
- allow installing and updating menuinst on Windows
- allow conda create with both --file and listed packages
- better handling of some nonexistent packages
- fix command line flags in conda package
- fix a bug in the ftp adapter

### 4.4.232 3.5.5 (2014-06-10)

- remove another instance pycosat version detection, which fails on Windows, see issue #761

#### 4.4.233 3.5.4 (2014-06-10)

- remove pycosat version detection, which fails on Windows, see issue #761

#### 4.4.234 3.5.3 (2014-06-09)

- fix conda update to correctly not install packages that are already up-to-date
- always fail with connection error in download
- the package resolution is now much faster and uses less memory
- add ssl_verify option in condarc to allow ignoring SSL certificate verification, see issue #737

#### 4.4.235 3.5.2 (2014-05-27)

- fix bug in activate.bat and deactivate.bat on Windows

#### 4.4.236 3.5.1 (2014-05-26)

- fix proxy support - conda now prompts for proxy username and password again
- fix activate.bat on Windows with spaces in the path
- update optional psutil dependency was updated to psutil 2.0 or higher

#### 4.4.237 3.5.0 (2014-05-15)

- replace use of urllib2 with requests. requests is now a hard dependency of conda.
- add ability to only allow system-wise specified channels
- hide binstar from output of conda info

#### 4.4.238 3.4.3 (2014-05-05)

- allow prefix replacement in binary files, see issue #710
- check if creating hard link is possible and otherwise copy, during install
- allow circular dependencies

#### 4.4.239 3.4.2 (2014-04-21)

- conda clean --lock: skip directories that don't exist, fixes #648
- fixed empty history file causing crash, issue #644
- remove timezone information from history file, fixes issue #651
- fix PackagesNotFound error for missing recursive dependencies
- change the default for adding cache from the local package cache - known is now the default and the option to use index metadata from the local package cache is --unknown
- add --alt-hint as a method to get an alternate form of a hint for unsatisfiable packages

- add conda package --ls-files to list files in a package
- add ability to pin specs in an environment. To pin a spec, add a file called pinned to the environment's conda-meta directory with the specs to pin. Pinned specs are always kept installed, unless the --no-pin flag is used.
- fix keyboard interrupting of external commands. Now keyboard interrupting conda build correctly removes the lock file
- add no_link ability to conda, see issue #678

### 4.4.240 3.4.1 (2014-04-07)

- always use a pkgs cache directory associated with an envs directory, even when using -p option with an arbitrary a prefix which is not inside an envs dir
- add setting of PYTHONHOME to conda info --system
- skip packages with bad metadata

### 4.4.241 3.4.0 (2014-04-02)

- added revision history to each environment:
  - conda list --revisions
  - conda install --revision
  - log is stored in conda-meta/history
- allow parsing pip-style requirement files with --file option and in command line arguments, e.g. conda install 'numpy>=1.7', issue #624
- fix error message for --file option when file does not exist
- allow DEFAULTS in CONDA_ENVS_PATH, which expands to the defaults settings, including the condarc file
- don't install a package with a feature (like mkl) unless it is specifically requested (i.e., that feature is already enabled in that environment)
- add ability to show channel URLs when displaying what is going to be downloaded by setting "show_channel_urls: True" in condarc
- fix the --quiet option
- skip packages that have dependencies that can't be found

### 4.4.242 3.3.2 (2014-03-24)

- fix the --file option
- check install arguments before fetching metadata
- fix a printing glitch with the progress bars
- give a better error message for conda clean with no arguments
- don't include unknown packages when searching another platform

### 4.4.243 3.3.1 (2014-03-19)

- Fix setting of PS1 in activate.
- Add conda update --all.
- Allow setting CONDARC=' ' to use no condarc.
- Add conda clean --packages.
- Don't include bin/conda, bin/activate, or bin/deactivate in conda package.

### 4.4.244 3.3.0 (2014-03-18)

- allow new package specification, i.e. ==, >=, >, <=, <, != separated by ',' for example: >=2.3,<3.0
- add ability to disable self update of conda, by setting "self_update: False" in .condarc
- Try installing packages using the old way of just installing the maximum versions of things first. This provides a major speedup of solving the package specifications in the cases where this scheme works.
- Don't include python=3.3 in the specs automatically for the Python 3 version of conda. This allows you to do "conda create -n env package" for a package that only has a Python 2 version without specifying "python=2". This change has no effect in Python 2.
- Automatically put symlinks to conda, activate, and deactivate in each environment on Unix.
- On Unix, activate and deactivate now remove the root environment from the PATH. This should prevent "bleed through" issues with commands not installed in the activated environment but that are installed in the root environment. If you have "setup.py develop" installed conda on Unix, you should run this command again, as the activate and deactivate scripts have changed.
- Begin work to support Python 3.4.
- Fix a bug in version comparison
- Fix usage of sys.stdout and sys.stderr in environments like pythonw on Windows where they are nonstandard file descriptors.

### 4.4.245 3.2.1 (2014-03-12)

- fix installing packages with irrational versions
- fix installation in the api
- use a logging handler to print the dots

### 4.4.246 3.2.0 (2014-03-11)

- print dots to the screen for progress
- move logic functions from resolve to logic module

### 4.4.247 3.2.0a1 (2014-03-07)

- conda now uses pseudo-boolean constraints in the SAT solver. This allows it to search for all versions at once, rather than only the latest (issue #491).
- Conda contains a brand new logic submodule for converting pseudo-boolean constraints into SAT clauses.

### 4.4.248 3.1.1 (2014-03-07)

- check if directory exists, fixed issue #591

### 4.4.249 3.1.0 (2014-03-07)

- local packages in cache are now added to the index, this may be disabled by using the --known option, which only makes conda use index metadata from the known remote channels
- add --use-index-cache option to enable using cache of channel index files
- fix ownership of files when installing as root on Linux
- conda search: add '.' symbol for extracted (cached) packages

### 4.4.250 3.0.6 (2014-02-20)

- fix 'conda update' taking build number into account

### 4.4.251 3.0.5 (2014-02-17)

- allow packages from create_default_packages to be overridden from the command line
- fixed typo install.py, issue #566
- try to prevent accidentally installing into a non-root conda environment

### 4.4.252 3.0.4 (2014-02-14)

- conda update: don't try to update packages that are already up-to-date

### 4.4.253 3.0.3 (2014-02-06)

- improve the speed of clean --lock
- some fixes to conda config
- more tests added
- choose the first solution rather than the last when there are more than one, since this is more likely to be the one you want.

### 4.4.254 3.0.2 (2014-02-03)

- fix detection of prefix being writable

### 4.4.255 3.0.1 (2014-01-31)

- bug: not having track_features in condarc now uses default again
- improved test suite
- remove numpy version being treated special in plan module
- if the post-link.(bat|sh) fails, don't treat it as though it installed, i.e. it is not added to conda-meta
- fix activate if CONDA_DEFAULT_ENV is invalid
- fix conda config --get to work with list keys again
- print the total download size
- fix a bug that was preventing conda from working in Python 3
- add ability to run pre-link script, issue #548

### 4.4.256 3.0.0 (2014-01-24)

- removed build, convert, index, and skeleton commands, which are now part of the conda-build project: https://github.com/conda/conda-build
- limited pip integration to `conda list`, that means `conda install` no longer calls `pip install` # !!!
- add ability to call sub-commands named 'conda-x'
- The -c flag to conda search is now shorthand for --channel, not –canonical (this is to be consistent with other conda commands)
- allow changing location of .condarc file using the CONDARC environment variable
- conda search now shows the channel that the package comes from
- conda search has a new --platform flag for searching for packages in other platforms.
- remove condarc warnings: issue #526#issuecomment-33195012

### 4.4.257 2.3.1 (2014-01-17)

- add ability create info/no_softlink
- add conda convert command to convert non-platform-dependent packages from one platform to another (experimental)
- unify create, install, and update code. This adds many features to create and update that were previously only available to install. A backwards incompatible change is that conda create -f now means --force, not –file.

### 4.4.258 2.3.0 (2014-01-16)

- automatically prepend http://conda.binstar.org/ (or the value of channel_alias in the .condarc file) to channels whenever the channel is not a URL or the word 'defaults or 'system'
- recipes made with the skeleton pypi command will use setuptools instead of distribute
- re-work the setuptools dependency and entry_point logic so that non console_script entry_points for packages with a dependency on setuptools will get correct build script with conda skeleton pypi
- add -m, --mkdir option to conda install
- add ability to disable soft-linking

### 4.4.259 2.2.8 (2014-01-06)

- add check for chrpath (on Linux) before build is started, see issue #469
- conda build: fixed ELF headers not being recognized on Python 3
- fixed issues: #467, #476

### 4.4.260 2.2.7 (2014-01-02)

- fixed bug in conda build related to lchmod not being available on all platforms

### 4.4.261 2.2.6 (2013-12-31)

- fix test section for automatic recipe creation from pypi using --build-recipe
- minor Py3k fixes for conda build on Linux
- copy symlinks as symlinks, issue #437
- fix explicit install (e.g. from output of `conda list -e`) in root env
- add pyyaml to the list of packages which can not be removed from root environment
- fixed minor issues: #365, #453

### 4.4.262 2.2.5 (2013-12-17)

- conda build: move broken packages to conda-bld/broken
- conda config: automatically add the 'defaults' channel
- conda build: improve error handling for invalid recipe directory
- add ability to set build string, issue #425
- fix LD_RUN_PATH not being set on Linux under Python 3, see issue #427, thanks peter1000

### 4.4.263 2.2.4 (2013-12-10)

- add support for execution with the -m switch (issue #398), i.e. you can execute conda also as: python -m conda
- add a deactivate script for windows
- conda build adds .pth-file when it encounters an egg (TODO)
- add ability to preserve egg directory when building using build/preserve_egg_dir: True
- allow track_features in ~/.condarc
- Allow arbitrary source, issue #405
- fixed minor issues: #393, #402, #409, #413

### 4.4.264 2.2.3 (2013-12-03)

- add "foreign mode", i.e. disallow install of certain packages when using a "foreign" Python, such as the system Python
- remove activate/deactivate from source tarball created by sdist.sh, in order to not overwrite activate script from virtualenvwrapper

### 4.4.265 2.2.2 (2013-11-27)

- remove ARCH environment variable for being able to change architecture
- add PKG_NAME, PKG_VERSION to environment when running build.sh, .-post-link.sh and .-pre-unlink.sh

### 4.4.266 2.2.1 (2013-11-15)

- minor fixes related to make conda pip installable
- generated conda meta-data missing 'files' key, fixed issue #357

### 4.4.267 2.2.0 (2013-11-14)

- add conda init command, to allow installing conda via pip
- fix prefix being replaced by placeholder after conda build on Unix
- add 'use_pip' to condarc configuration file
- fixed activate on Windows to set CONDA_DEFAULT_ENV
- allow setting "always_yes: True" in condarc file, which implies always using the --yes option whenever asked to proceed

### 4.4.268 2.1.0 (2013-11-07)

- fix rm_egg_dirs so that the .egg_info file can be a zip file
- improve integration with pip
  - conda list now shows pip installed packages
  - conda install will try to install via "pip install" if no conda package is available (unless --no-pip is provided)
  - conda build has a new --build-recipe option which will create a recipe (stored in /conda-recipes) from pypi then build a conda package (and install it)
  - pip list and pip install only happen if pip is installed
- enhance the locking mechanism so that conda can call itself in the same process.

### 4.4.269 2.0.4 (2013-11-04)

- ensure lowercase name when generating package info, fixed issue #329
- on Windows, handle the .nonadmin files

### 4.4.270 2.0.3 (2013-10-28)

- update bundle format
- fix bug when displaying packages to be downloaded (thanks Crystal)

### 4.4.271 2.0.2 (2013-10-27)

- add --index-cache option to clean command, see issue #321
- use RPATH (instead of RUNPATH) when building packages on Linux

### 4.4.272 2.0.1 (2013-10-23)

- add --no-prompt option to conda skeleton pypi
- add create_default_packages to condarc (and --no-default-packages option to create command)

### 4.4.273 2.0.0 (2013-10-01)

- added user/root mode and ability to soft-link across filesystems
- added create --clone option for copying local environments
- fixed behavior when installing into an environment which does not exist yet, i.e. an error occurs
- fixed install --no-deps option
- added --export option to list command
- allow building of packages in "user mode"
- regular environment locations now used for build and test
- add ability to disallow specification names

- add ability to read help messages from a file when install location is RO
- restore backwards compatibility of share/clone for conda-api
- add new conda bundle command and format
- pass ARCH environment variable to build scripts
- added progress bar to source download for conda build, issue #230
- added ability to use url instead of local file to conda install --file and conda create --file options

### 4.4.274 1.9.1 (2013-09-06)

- fix bug in new caching of repodata index

### 4.4.275 1.9.0 (2013-09-05)

- add caching of repodata index
- add activate command on Windows
- add conda package --which option, closes issue 163
- add ability to install file which contains multiple packages, issue 256
- move conda share functionality to conda package --share
- update documentation
- improve error messages when external dependencies are unavailable
- add implementation for issue 194: post-link or pre-unlink may append to a special file ${PREFIX}/.messages.txt for messages, which is display to the user's console after conda completes all actions
- add conda search --outdated option, which lists only installed packages for which newer versions are available
- fixed numerous Py3k issues, in particular with the build command

### 4.4.276 1.8.2 (2013-08-16)

- add conda build --check option
- add conda clean --lock option
- fixed error in recipe causing conda traceback, issue 158
- fixes conda build error in Python 3, issue 238
- improve error message when test command fails, as well as issue 229
- disable Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- simplified locking, in particular locking should never crash conda when files cannot be created due to permission problems

### 4.4.277 1.8.1 (2013-08-07)

- fixed conda update for no arguments, issue 237
- fix setting prefix before calling should_do_win_subprocess() part of issue 235
- add basic subversion support when building
- add --output option to conda build

### 4.4.278 1.8.0 (2013-07-31)

- add Python 3 support (thanks almarklein)
- add Mercurial support when building from source (thanks delicb)
- allow Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- add conda config command
- add conda clean command
- removed the conda pip command
- improve locking to be finer grained
- made activate/deactivate work with zsh (thanks to mika-fischer)
- allow conda build to take tarballs containing a recipe as arguments
- add PKG_CONFIG_PATH to build environment variables
- fix entry point scripts pointing to wrong python when building Python 3 packages
- allow source/sha1 in meta.yaml, issue 196
- more informative message when there are unsatisfiable package specifications
- ability to set the proxy urls in condarc
- conda build asks to upload to binstar. This can also be configured by changing binstar_upload in condarc.
- basic tab completion if the argcomplete package is installed and eval "$(register-python-argcomplete conda)" is added to the bash profile.

### 4.4.279 1.7.2 (2013-07-02)

- fixed conda update when packages include a post-link step which was caused by subprocess being lazily imported, fixed by 0d0b860
- improve error message when 'chrpath' or 'patch' is not installed and needed by build framework
- fixed sharing/cloning being broken (issue 179)
- add the string LOCKERROR to the conda lock error message

### 4.4.280 1.7.1 (2013-06-21)

- fix "executable" not being found on Windows when ending with .bat when launching application
- give a better error message from when a repository does not exist

### 4.4.281 1.7.0 (2013-06-20)

- allow ${PREFIX} in app_entry
- add binstar upload information after conda build finishes

### 4.4.282 1.7.0a2 (2013-06-20)

- add global conda lock file for only allowing one instance of conda to run at the same time
- add conda skeleton command to create recipes from PyPI
- add ability to run post-link and pre-unlink script

### 4.4.283 1.7.0a1 (2013-06-13)

- add ability to build conda packages from "recipes", using the conda build command, for some examples, see: https://github.com/ContinuumIO/conda-recipes
- fixed bug in conda install --force
- conda update command no longer uses anaconda as default package name
- add proxy support
- added application API to conda.api module
- add -c/–channel and --override-channels flags (issue 121).
- add default and system meta-channels, for use in .condarc and with -c (issue 122).
- fixed ability to install ipython=0.13.0 (issue 130)

### 4.4.284 1.6.0 (2013-06-05)

- update package command to reflect changes in repodata
- fixed refactoring bugs in share/clone
- warn when anaconda processes are running on install in Windows (should fix most permissions errors on Windows)

### 4.4.285 1.6.0rc2 (2013-05-31)

- conda with no arguments now prints help text (issue 111)
- don't allow removing conda from root environment
- conda update python does no longer update to Python 3, also ensure that conda itself is always installed into the root environment (issue 110)

### 4.4.286 1.6.0rc1 (2013-05-30)

- major internal refactoring
- use new "depends" key in repodata
- uses pycosat to solve constraints more efficiently
- add hard-linking on Windows
- fixed linking across filesystems (issue 103)
- add conda remove --features option
- added more tests, in particular for new dependency resolver
- add internal DSL to perform install actions
- add package size to download preview
- add conda install --force and --no-deps options
- fixed conda help command
- add conda remove --all option for removing entire environment
- fixed source activate on systems where sourcing a gives "bash" as $0
- add information about installed versions to conda search command
- removed known "locations"
- add output about installed packages when update and install do nothing
- changed default when prompted for y/n in CLI to yes

### 4.4.287 1.5.2 (2013-04-29)

- fixed issue 59: bad error message when pkgs dir is not writable

### 4.4.288 1.5.1 (2013-04-19)

- fixed issue 71 and (73 duplicate): not being able to install packages starting with conda (such as 'conda-api')
- fixed issue 69 (not being able to update Python / NumPy)
- fixed issue 76 (cannot install mkl on OSX)

#### 4.4.289 1.5.0 (2013-03-22)

- add conda share and clone commands
- add (hidden) --output-json option to clone, share and info commands to support the conda-api package
- add repo sub-directory type 'linux-armv6l'

#### 4.4.290 1.4.6 (2013-03-12)

- fixed channel selection (issue #56)

#### 4.4.291 1.4.5 (2013-03-11)

- fix issue #53 with install for meta packages
- add -q/–quiet option to update command

#### 4.4.292 1.4.4 (2013-03-09)

- use numpy 1.7 as default on all platforms

#### 4.4.293 1.4.3 (2013-03-09)

- fixed bug in conda.builder.share.clone_bundle()

#### 4.4.294 1.4.2 (2013-03-08)

- feature selection fix for update
- Windows: don't allow linking or unlinking python from the root environment because the file lock, see issue #42

#### 4.4.295 1.4.1 (2013-03-07)

- fix some feature selection bugs
- never exit in activate and deactivate
- improve help and error messages

#### 4.4.296 1.4.0 (2013-03-05)

- fixed conda pip NAME==VERSION
- added conda info --license option
- add source activate and deactivate commands
- rename the old activate and deactivate to link and unlink
- add ability for environments to track "features"

- add ability to distinguish conda build packages from Anaconda packages by adding a "file_hash" meta-data field in info/index.json
- add conda.builder.share module

### 4.4.297 1.3.5 (2013-02-05)

- fixed detecting untracked files on Windows
- removed backwards compatibility to conda 1.0 version

### 4.4.298 1.3.4 (2013-01-28)

- fixed conda installing itself into environments (issue #10)
- fixed non-existing channels being silently ignored (issue #12)
- fixed trailing slash in ~/.condarc file cause crash (issue #13)
- fixed conda list not working when ~/.condarc is missing (issue #14)
- fixed conda install not working for Python 2.6 environment (issue #17)
- added simple first cut implementation of remove command (issue #11)
- pip, build commands: only package up new untracked files
- allow a system-wide <sys.prefix>/.condarc (~/.condarc takes precedence)
- only add pro channel is no condarc file exists (and license is valid)

### 4.4.299 1.3.3 (2013-01-23)

- fix conda create not filtering channels correctly
- remove (hidden) --test and --testgui options

### 4.4.300 1.3.2 (2013-01-23)

- fix deactivation of packages with same build number note that conda upgrade did not suffer from this problem, as was using separate logic

### 4.4.301 1.3.1 (2013-01-22)

- fix bug in conda update not installing new dependencies

#### 4.4.302 1.3.0 (2013-01-22)

- added conda package command
- added conda index command
- added -c, --canonical option to list and search commands
- fixed conda --version on Windows
- add this changelog

#### 4.4.303 1.2.1 (2012-11-21)

- remove ambiguity from conda update command

#### 4.4.304 1.2.0 (2012-11-20)

- "conda upgrade" now updates from AnacondaCE to Anaconda (removed upgrade2pro
- add versioneer

#### 4.4.305 1.1.0 (2012-11-13)

- Many new features implemented by Bryan

#### 4.4.306 1.0.0 (2012-09-06)

- initial release

### 4.5 Glossary

#### 4.5.1 .condarc

The Conda Runtime Configuration file, an optional `.yaml` file that allows you to configure many aspects of conda, such as which channels it searches for packages, proxy settings, and environment directories. A `.condarc` file is not included by default, but it is automatically created in your home directory when you use the `conda config` command. The `.condarc` file can also be located in a root environment, in which case it overrides any `.condarc` in the home directory. For more information, see *Using the .condarc conda configuration file* and *Administering a multi-user conda installation*. Pronounced "conda r-c".

### 4.5.2 Activate/Deactivate environment

Conda commands used to switch or move between installed environments. The `conda activate` command prepends the path of your current environment to the PATH environment variable so that you do not need to type it each time. `deactivate` removes it. These commands might also execute activation and deactivation logic specified by some installed packages in your environment. Even when an environment is deactivated, you can still execute programs in that environment by specifying their paths directly, as in `~/anaconda/envs/envname/bin/program_name`. When an environment is activated, you can execute the program in that environment with just `program_name`.

> **Note**
>
> Replace `envname` with the name of the environment and replace `program_name` with the name of the program.

### 4.5.3 Anaconda

A downloadable, free, open-source, high-performance, and optimized Python and R distribution. Anaconda includes *conda*, conda-build, Python, and 250+ automatically installed, open-source scientific packages and their dependencies that have been tested to work well together, including SciPy, NumPy, and many others. Use the `conda install` command to easily install 7,500+ popular open-source packages for data science--including advanced and scientific analytics--from the Anaconda repository. Use the `conda` command to install thousands more open-source packages.

Because Anaconda is a Python distribution, it can make installing Python quick and easy even for new users.

Available for Windows, macOS, and Linux, all versions of Anaconda are supported by the community.

See also *Miniconda* and *conda*.

### 4.5.4 Anaconda.org

A web-based, repository hosting service in the cloud. Packages created locally can be published to the cloud to be shared with others. Anaconda.org is a public version of Anaconda Repository and was formerly known as Anaconda Cloud.

### 4.5.5 Anaconda Navigator

A desktop graphical user interface (GUI) included in all versions of Anaconda that allows you to easily manage conda packages, environments, channels, and notebooks without a command line interface (CLI). See more about Navigator.

### 4.5.6 Channels

The locations of the repositories where conda looks for packages. Channels may point to a Cloud repository or a private location on a remote or local repository that you or your organization created. The `conda channel` command has a default set of channels to search, beginning with https://repo.anaconda.com/pkgs/, which you may override, for example, to maintain a private or internal channel. These default channels are referred to in conda commands and in the `.condarc` file by the channel name "defaults."

### 4.5.7 conda

The package and environment manager program bundled with Anaconda that installs and updates conda packages and their dependencies. Conda also lets you easily switch between conda environments on your local computer.

### 4.5.8 conda environment

A folder or directory that contains a specific collection of conda packages and their dependencies, so they can be maintained and run separately without interference from each other. For example, you may use a conda environment for only Python 2 and Python 2 packages, maintain another conda environment with only Python 3 and Python 3 packages, and maintain another for R language packages. Environments can be created from:

- The Navigator GUI
- The command line
- An environment specification file with the name `your-environment-name.yml`

### 4.5.9 conda package

A compressed file that contains everything that a software program needs in order to be installed and run, so that you do not have to manually find and install each dependency separately. A conda package includes system-level libraries, Python or R language modules, executable programs, and other components. You manage conda packages with conda.

### 4.5.10 conda repository

A cloud-based repository that contains 7,500+ open-source certified packages that are easily installed locally with the `conda install` command. Anyone can access the repository from:

- The Navigator GUI
- A terminal using conda commands
- https://repo.anaconda.com/pkgs/

### 4.5.11 Metapackage

A metapackage is a very simple package that has at least a name and a version. It need not have any dependencies or build steps. *Metapackages* may list dependencies to several core, low-level libraries and may contain links to software files that are automatically downloaded when executed.

### 4.5.12 Miniconda

A free minimal installer for conda. Miniconda is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib, and a few others. Use the `conda install` command to install 7,500+ additional conda packages from the Anaconda repository.

Miniconda is a Python distribution that can make installing Python quick and easy even for new users.

See also *Anaconda* and *conda*.

### 4.5.13 Noarch package

A conda package that contains nothing specific to any system architecture, so it may be installed from any system. When conda searches for packages on any system in a channel, conda checks both the system-specific subdirectory, such as `linux-64`, and the `noarch` directory. Noarch is a contraction of "no architecture".

### 4.5.14 Package manager

A collection of software tools that automates the process of installing, updating, configuring, and removing computer programs for a computer's operating system. Also known as a package management system. Conda is a package manager.

### 4.5.15 Packages

Software files and information about the software, such as its name, the specific version, and a description, bundled into a file that can be installed and managed by a package manager.

### 4.5.16 Plugins

Plugins, sometimes referred to as add-ons or extensions, are software or modules that add new functions to a host program (*e.g.*, conda) without directly altering the host program itself. Amongst other uses, plugins support is utilized to enable third-party developers to extend an application, support easily adding new features, and to reduce the size of an application by not loading unused features.

### 4.5.17 Repository

Any storage location from which software assets may be retrieved and installed on a local computer. See also *Anaconda.org* and *conda repository*.

### 4.5.18 Silent mode installation

When installing Miniconda or Anaconda in silent mode, screen prompts are not shown on screen and default settings are automatically accepted.

## 4.6 Developer guide

### 4.6.1 Architecture

Conda is a complex system of many components and can be hard to understand for users and developers alike. The following C4 model based architecture diagrams should help in that regard. As a refresher, the C4 model tries to visualize complex software systems at different levels of detail, and explaining the functionality to different types of audience.

> **Note**
>
> These diagrams represent the state of conda at the time when the documentation was automatically build as part of the development process for conda 25.5.2.dev51 (Jul 03, 2025).

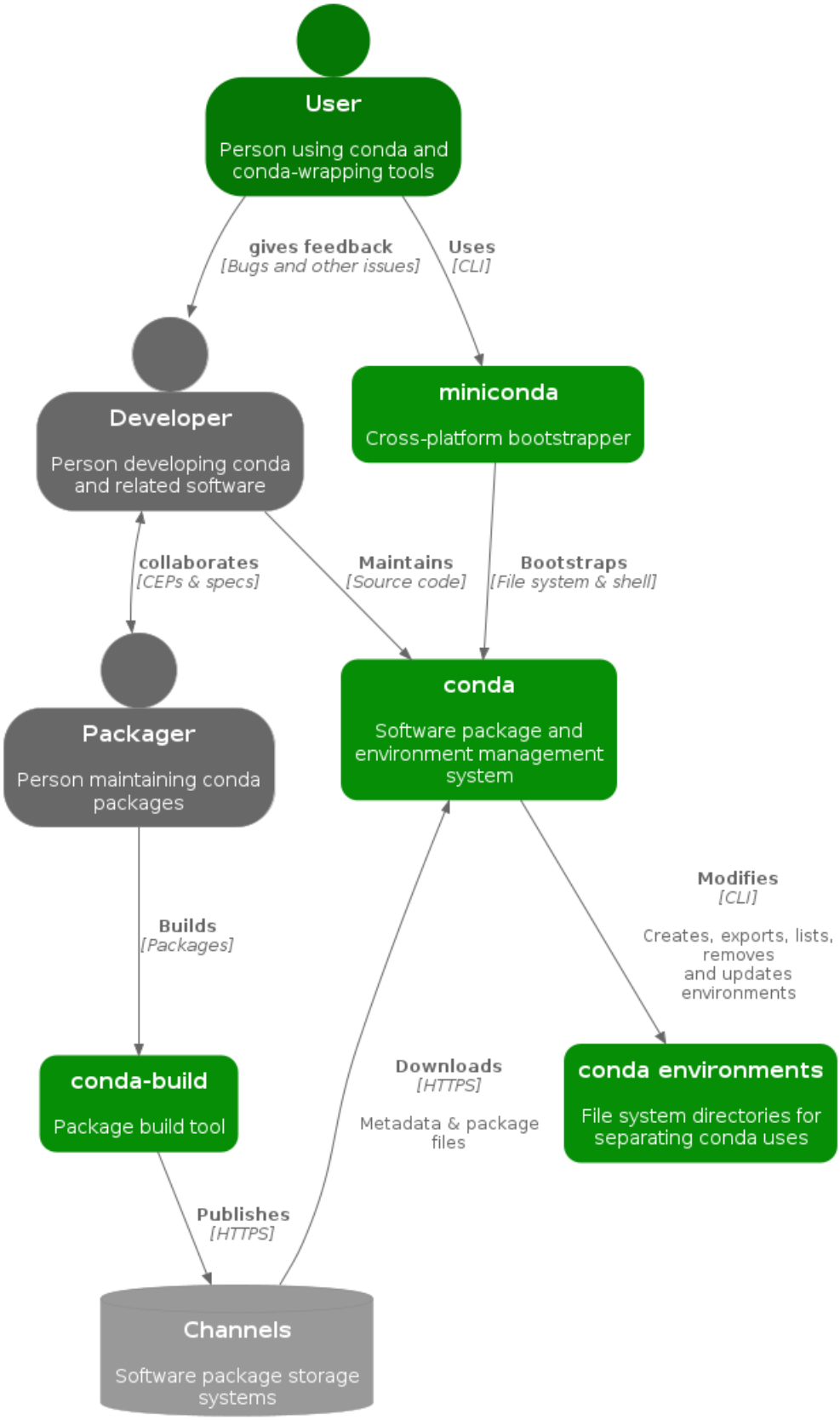
C4 stands for the for levels:

1. *Context*
2. *Container*
3. *Component*
4. *Code*

### Level 1: Context

This is the overview, 30,000 feet view on conda, to better understand how conda in the center of the diagram interacts with other systems and how users relate to it.

More information about how to interpret this diagram can be found in the C4 model documentation about the System Context diagram.


User
Person using conda and conda-wrapping tools
gives feedback
[Bugs and other issues]
Uses
[CLI]
Developer
Person developing conda and related software
miniconda
Cross-platform bootstrapper
collaborates
[CEPs & specs]
Maintains
[Source code]
Bootstraps
[File system & shell]
Packager
Person maintaining conda packages
conda
Software package and environment management system
Builds
[Packages]
Modifies
[CLI]
Creates, exports, lists, removes and updates environments
conda-build
Package build tool
Downloads
[HTTPS]
Metadata & package files
conda environments
File system directories for separating conda uses
Publishes
[HTTPS]
Channels
Software package storage systems

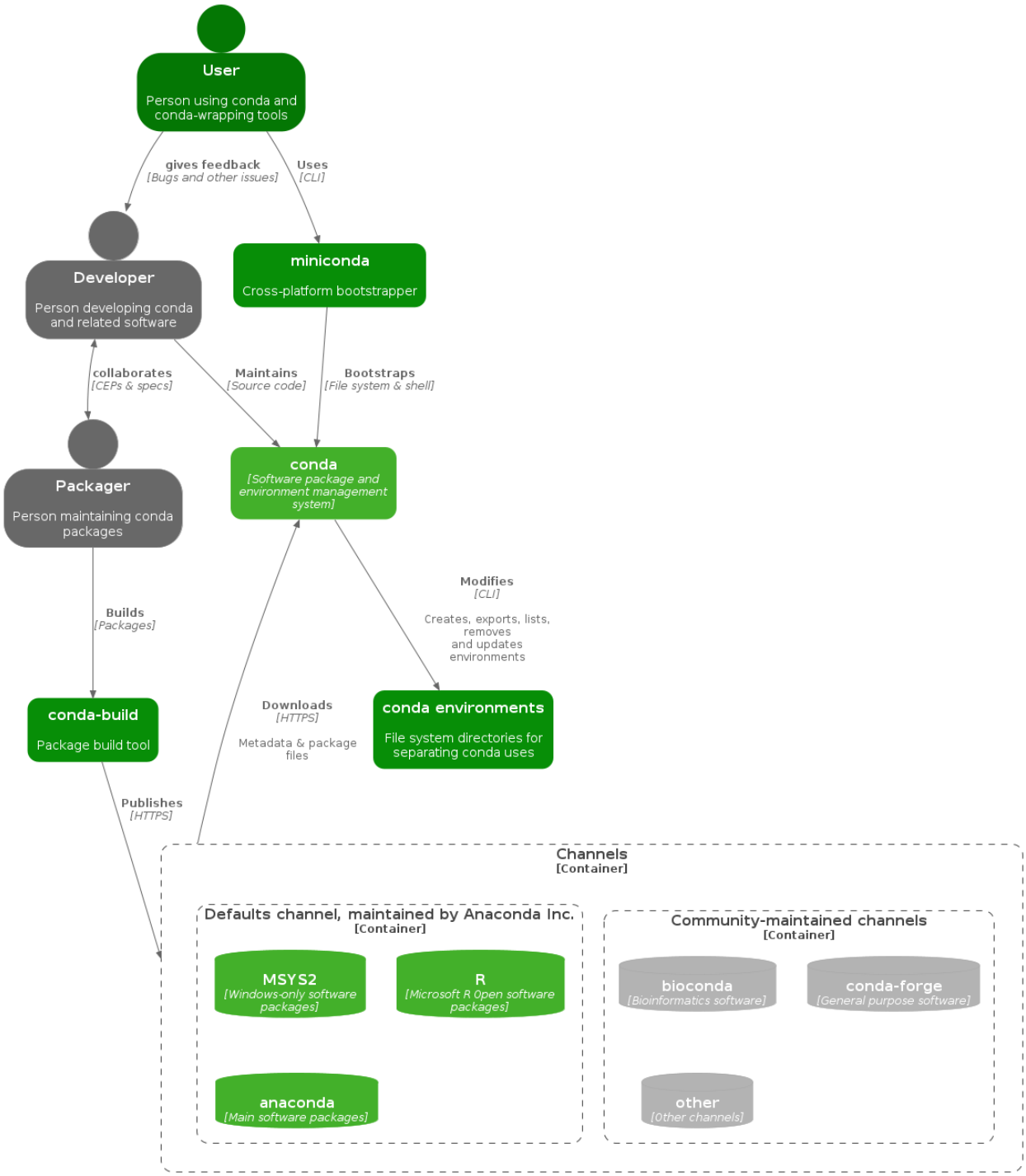
### Level 2: Container

This level is zooming in to conda on a system level, which was in the center of the Level 1 diagram, to show the high-level shape of the software architecture of and the various responsibilities in conda, including major technology choices and communication patterns between the various containers.

More information about how to interpret the following diagrams can be found in the C4 model documentation about the Container diagram.

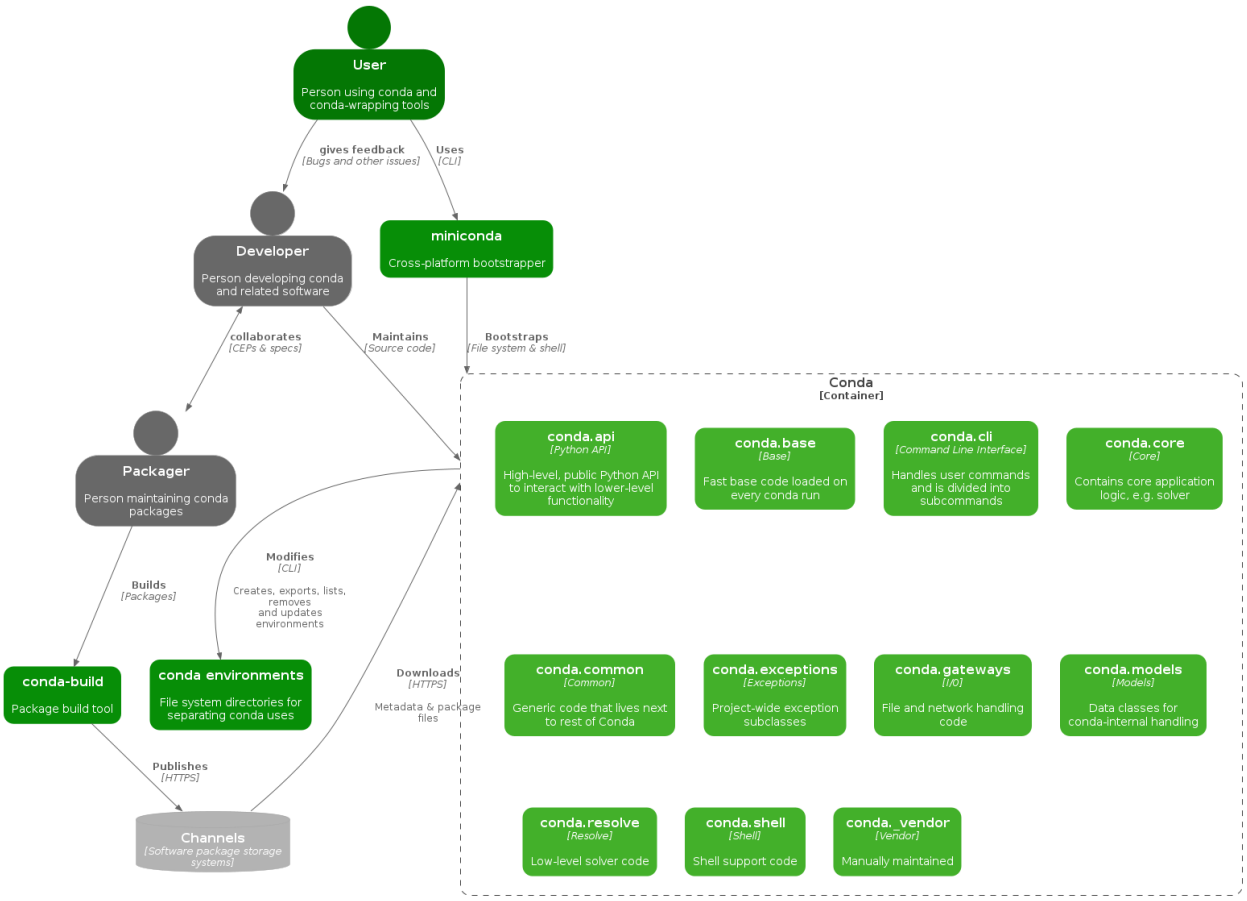
### Channels

The following diagram focuses on the channels container from the level 1 diagram.


User
Person using conda and conda-wrapping tools
gives feedback
[Bugs and other issues]
Uses
[CLI]
Developer
Person developing conda and related software
miniconda
Cross-platform bootstrapper
collaborates
[CEPs & specs]
Maintains
[Source code]
Bootstraps
[File system & shell]
Packager
Person maintaining conda packages
conda
[Software package and environment management system]
Builds
[Packages]
Modifies
[CLI]
Creates, exports, lists, removes and updates environments
conda-build
Package build tool
Downloads
[HTTPS]
Metadata & package files
conda environments
File system directories for separating conda uses
Publishes
[HTTPS]
Channels
[Container]
Defaults channel, maintained by Anaconda Inc.
[Container]
MSYS2
[Windows-only software packages]
R
[Microsoft R Open software packages]
anaconda
[Main software packages]
Community-maintained channels
[Container]
bioconda
[Bioinformatics software]
conda-forge
[General purpose software]
other
[Other channels]

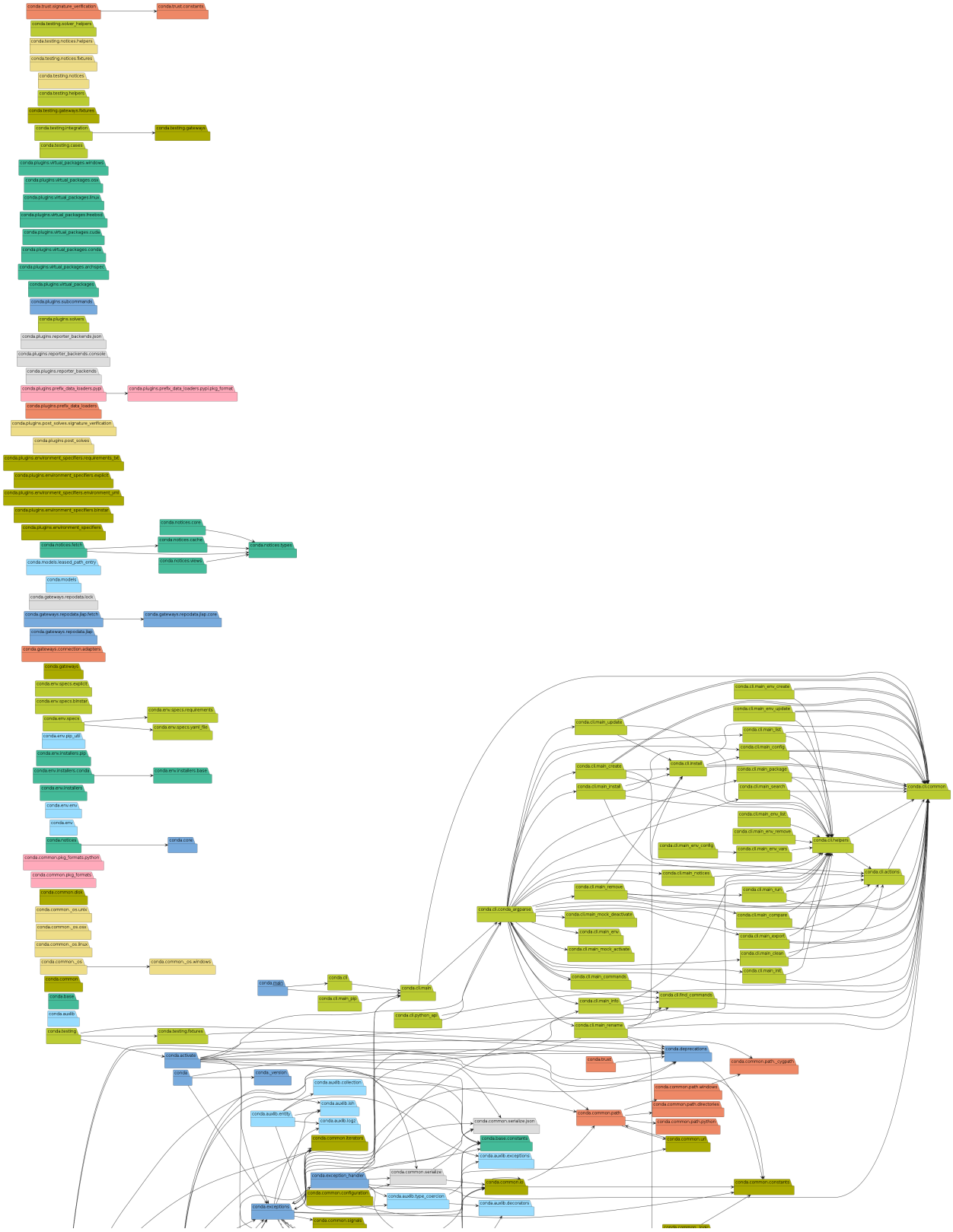
## Conda

The following diagram focuses on the conda container from the level 1 diagram.




## Level 3: Component

Yet another zoom-in, in which individual containers from Level 2 are decomposed to show major building blocks in conda and their interactions. Those building blocks are called components in the sense that they each have a higher function and relate to an identifiable responsibility and implementation details.

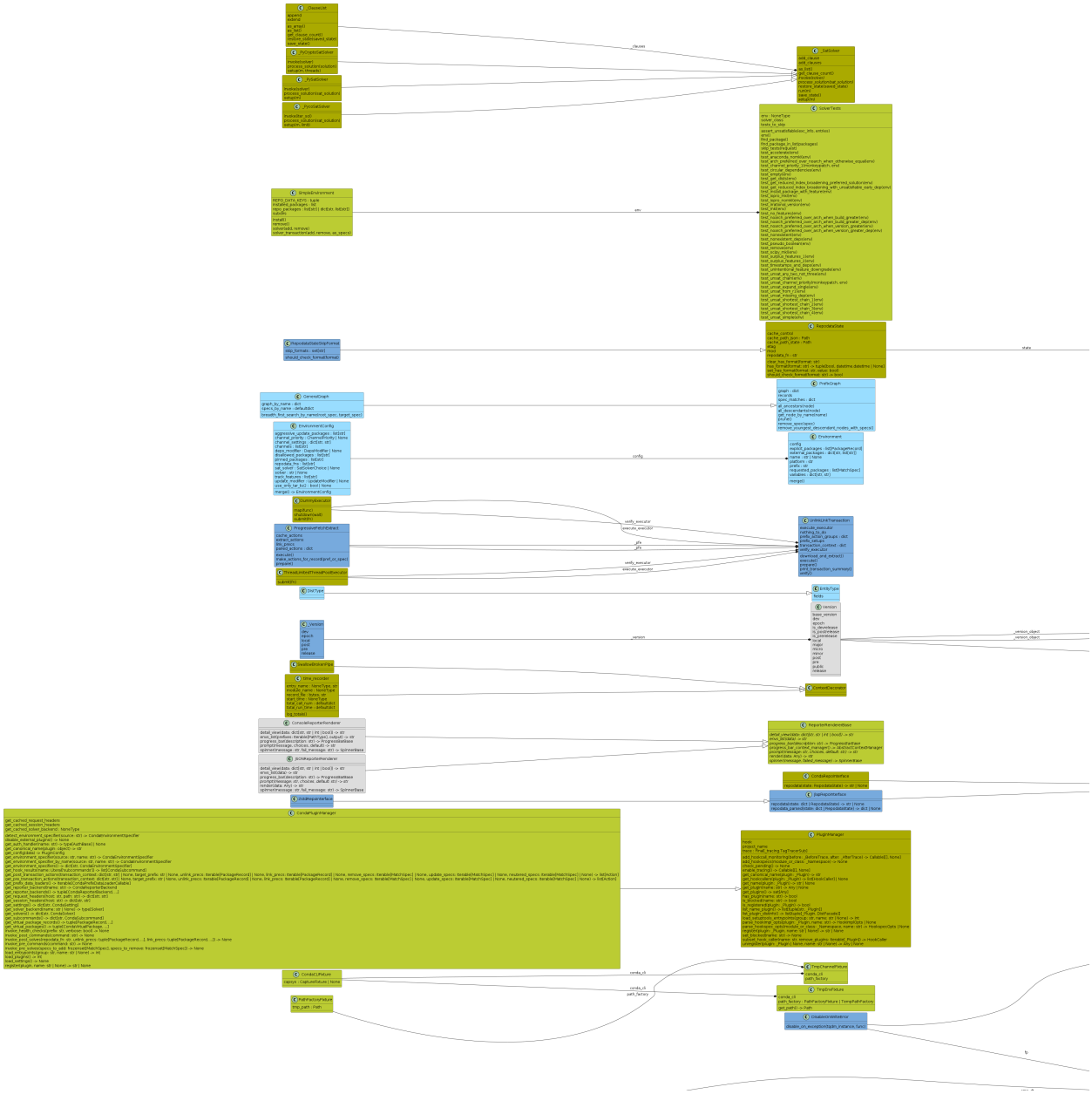



More information about how to interpret this diagram can be found in the C4 model documentation about the Component diagram.

### Level 4: Code

This part is auto-generated based on the current code and shows how the code is structured and how it interacts. For brevity this ignores a number of subsystems like the public API and exports modules, utility and vendor packages.

More information about how to interpret this diagram can be found in the C4 model documentation about the Code diagram.



### 4.6.2 Contributing to conda

Thank you for your interest in improving conda! Below, we describe how our development process works and how you can be a part of it.

*Already know how to contribute and need help setting up your development environment?* Read the development environment guide here

#### Hosted on GitHub

All development currently takes place on GitHub. This means we make extensive use of the project management tools they provide such as issues and projects.

#### Code of Conduct

When you decide to contribute to this project, it is important to adhere to our code of conduct, which is currently the NumFOCUS Code of Conduct. Please read it carefully.

#### Conda Contributor License Agreement

To begin contributing to this repository, you need to sign the Conda Contributor License Agreement (CLA). In case you're new to CLAs, this is a rather standard procedure for larger projects. Django and Python for example both use similar agreements.

Click here to sign the Conda Contributor License Agreement.

A record of prior signatories is kept in a separate repo in conda's GitHub organization.

#### Ways to contribute

Below are all the ways you can get involved in with conda.

#### Bug reports and feature requests

Bug reports and feature requests are always welcome. To file a new issue, head to the issue form.

It should be noted that `conda-build` issues need to be filed separately at its issue tracker.

For all other types of issues, please head to Anaconda.org's "Report a Bug" page. For even more information and documentation on everything related to Anaconda, head to the Support Center at Anaconda Nucleus.

Before submitting an issue via any of these channels, make sure to document it as well as possible and follow the submission guidelines (this makes everyone's job a lot easier!).

### Contributing your changes to conda

Here are the steps you need to take to contribute to conda:

1. Signup for a GitHub account (if you haven't already) and install Git on your system.
2. Sign the Conda Contributor License Agreement.
3. Fork the conda repository to your personal GitHub account by clicking the "Fork" button on https://github.com/conda/conda and follow GitHub's instructions.
4. Work on your proposed solution. *Visit this page if you need help getting your development environment setup*
5. When you are ready to submit a change, create a new pull request so that we can merge your changes to our repository.

### Issue sorting

Issue sorting is how we filter incoming issues and get them ready for active development. To see how this process works for this project, read "The Issue Sorting Process at conda".

*The project maintainers are currently not seeking help with issue sorting, but this may change in the future*

### Conda capitalization standards

1. Conda should be written in lowercase, whether in reference to the tool, ecosystem, packages, or organization.
2. References to the conda command should use code formatting (i.e. `conda`).
3. If the use of conda is not a command and if conda is at the beginning of a sentence, conda should be uppercase.

### Examples

### In sentences

Beginning a sentence:

- Conda is an open-source package and environment management system.
- `conda install` can be used to install packages.

Conda in the middle of a sentence:

- If a newer version of conda is available, you can use `conda update conda` to update to that version.
- You can find conda packages within conda channels. The `conda` command can search these channels.

#### In titles and headers

Titles and headers should use the same capitalization and formatting standards as sentences.

#### In links

Links should use the same capitalization conventions as sentences. Because the conda docs currently use reStructuredText (RST) as a markup language, and RST does not support nested inline markup, documentation writers should avoid using code backtick formatting inside links.

### 4.6.3 Development Environment

1. Clone the repo you just forked on GitHub to your local machine. Configure your repo to point to both "upstream" (the main conda repo) and your fork ("origin"). For detailed directions, see below:

   **Bash (macOS, Linux, Windows)**

   ```
   # choose the repository location
   # warning: not the location of an existing conda installation!
   $ CONDA_PROJECT_ROOT="$HOME/conda"

   # clone the project
   # replace `your-username` with your actual GitHub username
   $ git clone git@github.com:your-username/conda "$CONDA_PROJECT_ROOT"
   $ cd "$CONDA_PROJECT_ROOT"

   # set the `upstream` as the the main repository
   $ git remote add upstream git@github.com:conda/conda
   ```

   **cmd.exe (Windows)**

   ```
   # choose the repository location
   # warning: not the location of an existing conda installation!
   > set "CONDA_PROJECT_ROOT=%HOMEPATH%\conda"

   # clone the project
   # replace `your-username` with your actual GitHub username
   > git clone git@github.com:your-username/conda "%CONDA_PROJECT_ROOT%"
   > cd "%CONDA_PROJECT_ROOT%"

   # set the `upstream` as the main repository
   > git remote add upstream git@github.com:conda/conda
   ```

2. One option is to create a local development environment and activate that environment

   **Bash (macOS, Linux, Windows)**

   ```
   $ source ./dev/start
   ```

   **cmd.exe (Windows)**

   ```
   > .\dev\start.bat
   ```

This command will create a project-specific base environment (see `devenv` in your repo directory after running this command). If the base environment already exists this command will simply activate the already-created `devenv` environment.

To be sure that the conda code being interpreted is the code in the project directory, look at the value of `conda location:` in the output of `conda info --all`.

3. Alternatively, for Linux development only, you can use the same Docker image the CI pipelines use. Note that you can run this from all three operating systems! We are using `docker compose`, which provides three actions for you:

   - `unit-tests`: Run all unit tests.
   - `integration-tests`: Run all integration tests.
   - `interactive`: You are dropped in a pre-initialized Bash session, where you can run all your `pytest` commands as required.

   Use them with `docker compose run <action>`. For example:

   **Any shell (macOS, Linux, Windows)**

```
$ docker compose run unit-tests
```

   This builds the same Docker image as used in continuous integration from the Github Container Registry and starts `bash` with the conda development mode already enabled.

   By default, it will use Miniconda-based, Python 3.9 installation configured for the `defaults` channel. You can customize this with two environment variables:

   - `CONDA_DOCKER_PYTHON`: `major.minor` value; e.g. `3.11`.
   - `CONDA_DOCKER_DEFAULT_CHANNEL`: either `defaults` or `conda-forge`

   For example, if you need a conda-forge based 3.12 image:

   **Bash (macOS, Linux, Windows)**

```
$ CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose␣
↪build --no-cache
# --- in some systems you might also need to re-supply the same values as CLI flags:
# CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose␣
↪build --no-cache --build-arg python_version=3.12 --build-arg default_
↪channel=conda-forge
$ CONDA_DOCKER_PYTHON=3.12 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose␣
↪run interactive
```

   **cmd.exe (Windows)**

```
> set CONDA_DOCKER_PYTHON=3.12
> set CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge
> docker compose build --no-cache
> docker compose run interactive
> set "CONDA_DOCKER_PYTHON="
> set "CONDA_DOCKER_DEFAULT_CHANNEL="
```

   The `conda` repository will be mounted to `/opt/conda-src`, so all changes done in your editor will be reflected live while the Docker container is running.

### Static Code Analysis

This project is configured with pre-commit to automatically run linting and other static code analysis on every commit. Running these tools prior to the PR/code review process helps in two ways:

1. it helps *you* by automating the nitpicky process of identifying and correcting code style/quality issues
2. it helps *us* where during code review we can focus on the substance of your contribution

Feel free to read up on everything pre-commit related in their docs but we've included the gist of what you need to get started below:

**Bash (macOS, Linux, Windows)**

```
# reuse the development environment created above
$ source ./dev/start
# or start the Docker image in interactive mode
# $ docker compose run interactive

# install pre-commit hooks for conda
$ cd "$CONDA_PROJECT_ROOT"
$ pre-commit install

# manually running pre-commit on current changes
# note: by default pre-commit only runs on staged files
$ pre-commit run

# automatically running pre-commit during commit
$ git commit
```

**cmd.exe (Windows)**

```
:: reuse the development environment created above
> .\dev\start.bat
:: or start the Docker image in interactive mode
:: > docker compose run interactive

:: install pre-commit hooks for conda
> cd "%CONDA_PROJECT_ROOT%"
> pre-commit install

:: manually running pre-commit on current changes
:: note: by default pre-commit only runs on staged files
> pre-commit run

:: automatically running pre-commit during commit
> git commit
```

Beware that some of the tools run by pre-commit can potentially modify the code (see black, blacken-docs, and darker). If pre-commit detects that any files were modified it will terminate the commit giving you the opportunity to review the code before committing again.

Strictly speaking using pre-commit on your local machine for commits is optional (if you don't install pre-commit you will still be able to commit normally). But once you open a PR to contribue your changes, pre-commit will be automatically run at which point any errors that occur will need to be addressed prior to proceeding.

### Testing

We use pytest to run our test suite. Please consult pytest's docs for detailed instructions but generally speaking all you need is the following:

**Bash (macOS, Linux, Windows)**

```
# reuse the development environment created above
$ source ./dev/start
# or start the Docker image in interactive mode
# $ docker compose run interactive

# run conda's unit tests using GNU make
$ make unit

# or alternately with pytest
$ pytest --cov -m "not integration" conda tests

# or you can use pytest to focus on one specific test
$ pytest --cov tests/test_create.py -k create_install_update_remove_smoketest
```

**cmd.exe (Windows)**

```
:: reuse the development environment created above
> .\dev\start.bat
:: or start the Docker image in interactive mode
:: > docker compose run interactive

:: run conda's unit tests with pytest
> pytest --cov -m "not integration" conda tests

:: or you can use pytest to focus on one specific test
> pytest --cov tests\test_create.py -k create_install_update_remove_smoketest
```

If you are not measuring code coverage, `pytest` can be run without the `--cov` option. The `docker compose` tests pass `--cov`.

Note: Some integration tests require you build a package with conda-build beforehand. This is taking care of if you run `docker compose run integration-tests`, but you need to do it manually in other modes:

**Bash (macOS, Linux, Windows)**

```
$ conda install conda-build
$ conda-build tests/test-recipes/activate_deactivate_package tests/test-recipes/pre_link_
↪messages_package
```

Check `dev/linux/integration.sh` and `dev\windows\integration.bat` for more details.

### 4.6.4 Deep dives

This section contains a series of deep dives into particularly complex parts of conda.

#### `conda install`

In this document we will explore what happens in Conda from the moment a user types their installation command until the process is finished successfully. For the sake of completeness, we will consider the following situation:

- The user is running commands on a Linux x64 machine with a working installation of Miniconda.
- This means we have a `base` environment with `conda`, `python`, and their dependencies.
- The `base` environment is already preactivated for Bash. For more details on activation, check *conda init and conda activate*.

Ok, so… what happens when you run `conda install numpy`? Roughly, these steps:

1. Command line interface
    - `argparse` parsers
    - Environment variables
    - Configuration files
    - Context initialization
    - Delegation of the task
2. Fetching the index
    - Retrieving all the channels and platforms
    - A note on channel priorities
3. Solving the install request
    - Requested packages + prefix state = list of specs
    - Index reduction (sometimes)
    - Running the solver
    - Post-processing the list of packages
4. Generating the transaction and the corresponding actions
5. Download and extraction
6. Integrity verification
7. Linking and unlinking files
8. Post-linking and post-activation tasks

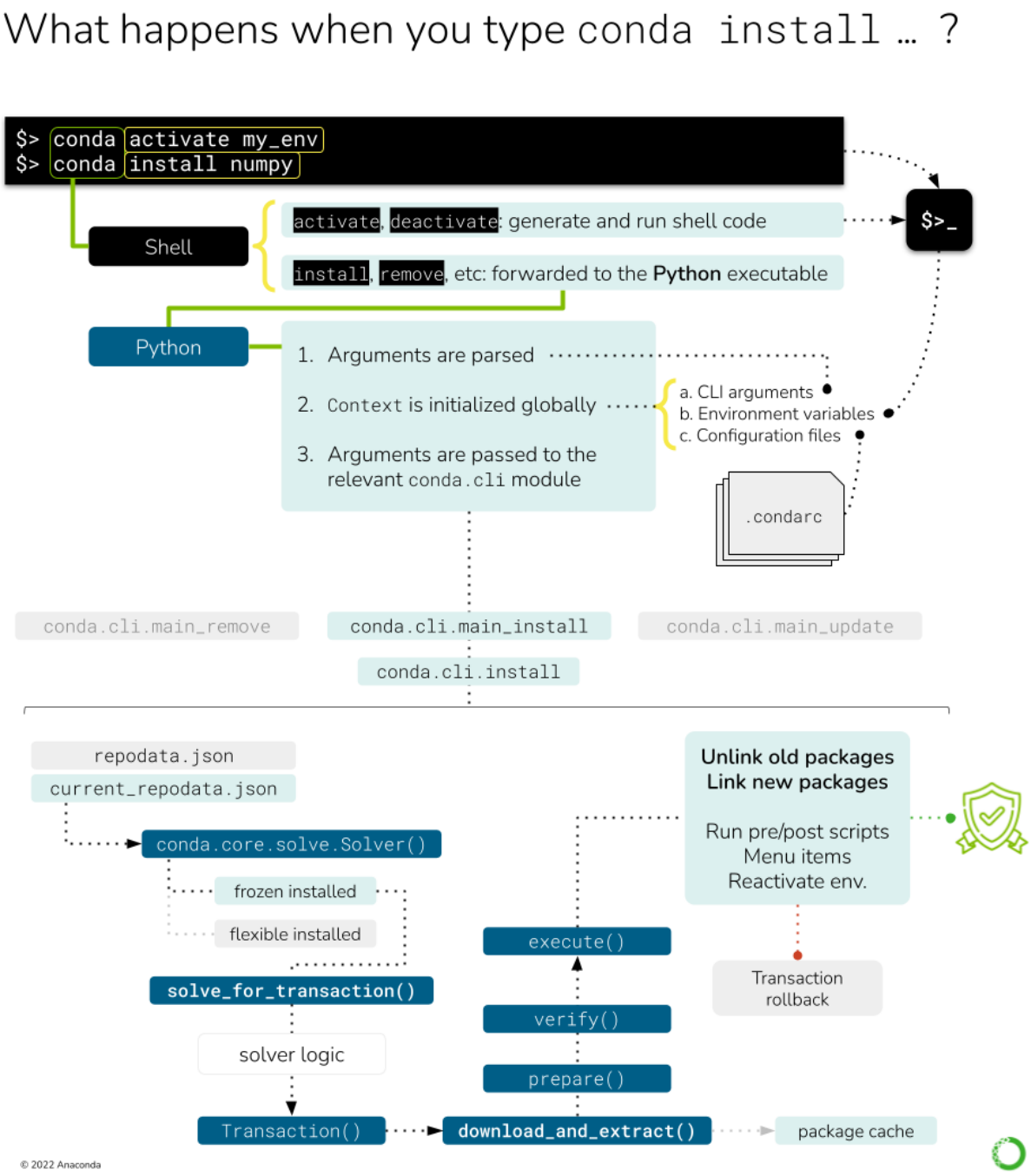



Fig. 1: This figure shows the different processes and objects involved in handling a simple `conda install` command.

### Command line interface

First, a quick note on an implementation detail that might be not obvious.

When you type `conda install numpy` in your terminal, Bash takes those three words and looks for a `conda` command to pass a list of arguments `['conda', 'install', 'numpy']`. Before finding the `conda` executable located at `CONDA_HOME/condabin`, it probably finds the shell function defined here. This shell function runs the activation/deactivation logic on the shell if requested, or delegates over to the actual Python entry-points otherwise. This part of the logic can be found in `conda.shell`.

Once we are running the Python entry-point, we are in the `conda.cli` realm. The function called by the entry point is `conda.cli.main:main()`. Here, another check is done for `shell.*` subcommands, which generate the shell initializers you see in `~/.bashrc` and others. If you are curious where this happens, it's `conda.activate`.

Since our command is `conda install ...`, we still need to arrive somewhere else. You will notice that the rest of the logic is delegated to `conda.cli.main:_main()`, which will invoke the parser generators, initialize the context and loggers, and, eventually, pass the argument list over to the corresponding command function. These four steps are implemented in four functions/classes:

1. `conda.cli.conda_argparse:generate_parser()`: This uses `argparse` to generate the CLI. Each subcommand is initialized in separate functions. Note that the command line options are not generated dynamically from the `Context` object, but annotated manually. If this is needed (e.g. `--repodata-fn` is exposed in `Context.repodata_fn`), the `dest` variable of each CLI option should match the target attribute in the context object.
2. `conda.base.context.Context`: This object stores the configuration options in `conda` and will be initialized taking into account, among other things, the arguments parsed in the step above. This is covered in more detail in a separate deep dive: *conda config and context*.
3. `conda.gateways.logging:initialize_logging()`: Not too exciting and easy to follow. This part of the code base is more or less self-explanatory.
4. `conda.cli.conda_argparse:do_call()`: The argument parsing will populate a `func` value that contains the import path to the function responsible for that subcommand. For example, `conda install` is taken care of by `conda.cli.main_install`. By design, all the modules reported by `func` must contain an `execute()` function that implements the command logic. `execute()` takes the parsed arguments and the parser itself as arguments. For example, in the case of `conda install`, `execute()` only redirects to a certain mode in `conda.cli.install:install()`.

Let's go take a look at that module now. `conda.cli.install:install()` implements the logic behind `conda create`, `conda install`, `conda update` and `conda remove`. In essence, they all deal with the same task: changing which packages are present in an environment. If you go and read that function, you will see there are several lines of code handling diverse situations (new environments, clones, etc.) before we arrive to the next section. We will not discuss them here, but feel free to explore that section. It's mostly ensuring that the destination prefix exists, whether we are creating a new environment and massaging some command line flags that would allow us to skip the solver (e.g. `--clone`).

**More information on environments**

Check the concepts for *Environments*.

### Fetching the index

At this point, we are ready to start doing some work! All of the previous code was telling us what to do, and now we know. We want `conda` to *install* `numpy` on our `base` environment. The first thing we need to know is where we can find packages with the name `numpy`. The answer is… the channels!

Users download packages from `conda` channels. These are normally hosted at `anaconda.org`. A channel is essentially a directory structure with these elements:

```
<channel>
├── channeldata.json
├── index.html
├── <platform> (e.g. linux-64)
│   ├── current_repodata.json
│   ├── current_repodata.json.bz2
│   ├── index.html
│   ├── repodata.json
│   ├── repodata.json.bz2
│   ├── repodata_from_packages.json
│   └── repodata_from_packages.json.bz2
└── noarch
    ├── current_repodata.json
    ├── current_repodata.json.bz2
    ├── index.html
    ├── repodata.json
    ├── repodata.json.bz2
    ├── repodata_from_packages.json
    └── repodata_from_packages.json.bz2
```

**More info on Channels**

You can find some more user-oriented notes on Channels at *What is a "channel"?* and *Repository structure and index*. If you are interested in more technical details, check the corresponding documentation pages at conda-build.

The important bits are:

- A channel contains one or more platform-specific directories (`linux-64`, `osx-64`, etc.), plus a platform-agnostic directory called `noarch`. In `conda` jargon, these are also referred to as channel *subdirs*. Officially, the `noarch` subdirectory is enough to make it a `conda` channel; e.g. no platform subdirectory is necessary.
- Each *subdir* contains at least a `repodata.json` file: a gigantic dictionary with *all* the metadata for each package available on that platform.
- In most cases, the same subdirs also contain the `*.tar.bz2` files for each of the published packages. This is what `conda` downloads and extracts once solving is complete. The anatomy of these files is well defined, both in content and naming structure. See *What is a package?*, *Package metadata* and/or Package naming conventions for more details.

Additionally, the channel's main directory might contain a `channeldata.json` file, with channel-wide metadata (this is not specific per platform). Not all channels include this, and in general it is not currently something that is commonly utilized.

Since conda's philosophy is to keep all packages ever published around for reproducibility, `repodata.json` is always growing, which presents a problem both for the download itself and the solver engine. To reduce download times and

bandwidth usage, `repodata.json` is also served as a BZIP2 compressed file, `repodata.json.bz2`. This is what most `conda` clients end up downloading.

> **Note on 'current_repodata.json'**
>
> More *repodatas* variations can be found in some channels, but they are always reduced versions of the main one for the sake of performance. For example, `current_repodata.json` only contains the most recent version of each package, plus their dependencies. The rationale behind this optimization trick can be found here.

So, in essence, fetching the channel information means it can be expressed in pseudo-code like this:

```
platform = {}
noarch = {}
for channel in reversed(context.channels):
    platform_repodata = fetch_extract_and_read(
        channel.full_url / context.subdir / "repodata.json.bz2"
    )
    platform.update(platform_repodata)
    noarch_repodata = fetch_extract_and_read(
        channel.full_url / "noarch" / "repodata.json.bz2"
    )
    noarch.update(noarch_repodata)
```

Note that these dictionaries are keyed by *filename*, so higher priority channels will overwrite entries with the exact same filename (e.g. `numpy-1.19-py36h87ha43_0.tar.bz2`). If they don't have the same *filename* (e.g., same version and build number but different hash), this ambiguity will be resolved later in the solver, taking into account the channel priority mode.

In this example, `context.channels` has been populated through different, cascading mechanisms:

- The default settings as found in `~/.condarc` or equivalent.
- The `CONDA_CHANNELS` environment variable (rare usage).
- The command-line flags, such as `-c <channel>`, `--use-local` or `--override-channels`.
- The channels present in a command-line *spec*. Remember that users can say `channel::numpy` instead of simply `numpy` to require that numpy comes from that specific channel. That means that the repodata for such channel needs to be fetched, too!

The items in `context.channels` are supposed to be `conda.models.channels.Channel` objects, but the Solver API also allows strings that refer to their name, alias or full URL. In that case, you can use `Channel` objects to parse and retrieve the full URL for each subdir using the `Channel.urls()` method. Several helper functions can be found in `conda.core.index`, if needed.

Sadly, `fetch_extract_and_read()` does not exist as such, but as a combination of objects. The main driving function is actually `get_index()`, which passes the channel URLs to `fetch_index`, a wrapper that delegates directly to `conda.core.subdir_data.SubdirData` objects. This object implements caching, authentication, proxies and other things that complicate the simple idea of "just download the file, please". Most of the logic is in `SubdirData._load()`, which ends up calling `conda.core.subdir_data.fetch_repodata_remote_request()` to process the request. Finally, `SubdirData._process_raw_repodata_str()` does the parsing and loading.

Internally, the `SubdirData` stores all the package metadata as a list of `PackageRecord` objects. Its main usage is via `.query()` (one result at a time) or `.query_all()` (all possible matches). These `.query*` methods accept spec strings (e.g. `numpy =1.14`), `MatchSpec` and `PackageRecord` instances. Alternatively, if you want *all* records with no queries, use `SubdirData.iter_records()`.

**Tricks to reduce the size of the index**

conda supports the notion of trying with different versions of the index in an effort to minimize the solution space. A smaller index means a faster search, after all! The default logic starts with `current_repodata.json` files in the channel, which contain only the latest versions of each package plus their dependencies. If that fails, then the full `repodata.json` is used. This happens *before* the `Solver` is even invoked.

The second trick is done within the classic solver logic (pycosat): an informed index reduction. In essence, the index (whether it's `current_repodata.json` or full `repodata.json`) is pruned by the solver, trying to keep only the parts that it anticipates will be needed. More details can be found on the `get_reduced_index` function. Interestingly, this optimization step also takes longer the bigger the index gets.

### Channel priorities

`context.channels` returns an `IndexedSet` of `Channel` objects; essentially a list of unique items. The different channels in this list can have overlapping or even conflicting information for the same package name. For example, `defaults` and `conda-forge` will for sure contain packages that fullfil the `conda install numpy` request. Which one is chosen by conda in this case? It depends on the `context.channel_priority` setting: From the help message:

> Accepts values of 'strict', 'flexible', and 'disabled'. The default value is 'flexible'. With strict channel priority, packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel. With flexible channel priority, the solver may reach into lower priority channels to fulfill dependencies, rather than raising an unsatisfiable error. With channel priority disabled, package version takes precedence, and the configured priority of channels is used only to break ties.

In practice, `channel_priority=strict` is often the recommended setting for most users. It's faster to solve and causes fewer problems down the line. Check more details *here*.

### Solving the install request

At this point, we can start asking the solver things. After all, we have loaded the channels into our index, building the catalog of available packages and versions we can install. We also have the command line instructions and configurations needed to customize the solver request. So, let's just do it: "Solver, please install numpy on this prefix using these channels as package sources".

The details are complicated, but in essence, the `Solver` will:

1. Express the requested packages, command line options and prefix state as `MatchSpec` objects
2. Query the index for the best possible match that satisfy those constraints
3. Return a list of `PackageRecord` objects

The full details are covered in *Solvers* if you are curious. Just keep in mind that point (1) is conda-specific, while (2) can be tackled, in principle, by any SAT solver.

### Generating the transaction and the corresponding actions

The Solver API defines three public methods:

- `.solve_final_state()`: this is the core function, described in the section above. Given some input state, it returns an `IndexedSet` of `PackageRecord` objects that reflect what the final state of the environment should look like. This is the largest method, and its details are fully covered *here*.
- `.solve_for_diff()`: this method takes the final state and diffs it with the current state of the environment, discovering which old records need to be removed, and which ones need to be added.
- `.solve_for_transaction()`: this method takes the diff and creates a `Transaction` object for this operation. This is what the main CLI logic expects back from the solver.

So what is a `Transaction` object and why is it needed? Transactional actions were introduced in conda 4.3. They seem to be the last iteration of a set of changes designed to check whether `conda` would be able to download and link the needed packages (e.g. check that there is enough space on disk, whether the user has enough permissions for the target paths, etc.). For more info, refer to PRs #3571, #3301, and #3034.

The transaction is essentially a set of `action` objects. Each action is allowed to run some checks to determine whether it can be executed successfully. If that's not the case, the failed checks will signal the parent transaction that the whole operation needs to be aborted and rolled back to leave things in the state they were before running that `conda` command. It is also responsible for some of the messages you will see in the CLI output, like the reports of what will be installed, updated or removed.

**Transactions and parallelism**

Since the transaction object knows about all the actions that need to happen, it also enables parallelism for verifying, downloading and (un)linking tasks. The level of parallelism can be changed through the following `context` settings:

- `default_threads`
- `verify_threads`
- `execute_threads`
- `repodata_threads`
- `fetch_threads`

There's only one class of transaction in `conda`: `UnlinkLinkTransaction`. It only accepts one input parameter: a list of `PrefixSetup` objects, which are just `namedtuple` objects with the following fields. These are populated by `Solver.solve_for_transaction` after running `Solver.solve_for_diff`:

- `target_prefix`: the environment path the command is running on.
- `unlink_precs`: `PackageRecord` objects that need to be unlinked (removed).
- `link_precs`: `PackageRecord` objects that need to be linked (added).
- `remove_specs`: `MatchSpec` objects that need to be marked as removed in the history (the user asked for these packages to be uninstalled).
- `update_specs`: `MatchSpec` objects that need to be marked as added in the history (the user asked for these packages to be installed or updated).
- `neutered_specs`: `MatchSpec` objects that were already in history but had to be relaxed in order to avoid solving conflicts.

Whatever happens after instantiation depends on the content of these `PrefixSetup` objects. Sometimes, the transaction results in no actions (see the `nothing_to_do` property) because the request asked by the user is already fulfilled by the current state of the environment.

However, most of the time the transaction will involve a number of actions. This is done via two public methods:

- `download_and_extract()`: essentially a forwarder to instantiate and call `ProgressiveFetchExtract`, responsible for deciding which `PackageRecords` need to be downloaded and extracted to the packages cache.
- `execute()`: the core logic is layed out here. It involves preparing, verifying and performing the rest of the actions. Among others:
  - Unlinking packages (removing a package from the environment)
  - Linking (adding a package to the environment)
  - Compiling bytecode (generating the `pyc` counterpart for each `py` module)
  - Adding entry points (generate command line executables for the configured functions)
  - Adding the JSON records (for each package, a JSON file is added to `conda-meta/`)
  - Make menu items (create shortcuts for packages featuring a JSON file under `Menu/`)
  - Remove menu items (remove the shortcuts created by that package)

It's important to notice that download and extraction happen separately from all the other actions. This separation is important and core to the idea of what a `conda` environment is. Essentially, when you create a new `conda` environment, you are not necessarily *copying* files over to the target prefix location. Instead, `conda` maintains a cache of every package ever downloaded to disk (both the tarball and the extracted contents). To save space and speed up environment creation and deletion, files are not copied over, but instead they are linked (usually via a hardlink). That's why these two tasks are separated in the transaction logic: you don't need to download and extract packages that are already in the cache; you only need to link them!

**Transactions also drive reports**

The type and number of actions can also be calculated by `_make_legacy_action_groups()`, which returns a list of *action groups* (one per `PrefixSetup`). Each action group is a just a dictionary following this specification:

```
{
  "FETCH": Iterable[PackageRecord],  # estimated by `ProgressiveFetchExtract`
  "PREFIX": str,
  "UNLINK": Iterable[PackageRecord],
  "LINK: Iterable[PackageRecord],
}
```

These simpler action groups are only used for reporting, either via a processed text report (via `print_transaction_summary`) or just the raw JSON (via `stdout_json_success`). As you can see, they do not know anything about other types of tasks.

### Download and extraction

conda maintains a cache of downloaded tarballs and their extracted contents to save disk space and improve the performance of environment modifications. This requires some code to check whether a given `PackageRecord` is already present in the cache, and, if it's not, how to download the tarball and extract its contents in a performant way. This is all handled by the `ProgressiveFetchExtract` class, which can instantiate up to two `Action` objects for each passed `PackageRecord`:

- `CacheUrlAction`: downloads (if remote) or copies (if local) a tarball to the cache location.
- `ExtractPackageAction`: extracts the contents of the tarball.

These two actions only take place *if* the package is not in cache yet and if it has already been extracted, respectively. They can also revert the changes if the transaction is aborted (either due to an error or because the user pressed Ctrl+C).

### Populating the prefix

When all the necessary packages have been downloaded and extracted to the cache, it is time to start populating the prefix with the needed files. This means we need to:

1. For each package that needs to be unlinked, run the pre-unlink logic (`deactivate` and `pre-unlink` scripts, as well as shortcut removal, if needed) and then unlink the package files.
2. For each package that needs to be linked, create the links and run the post-link logic (`post-link` and `activate` scripts, as well as creating the shortcuts, if needed).

   Note that when you are updating a package version, you are actually removing the installed version entirely and then adding the new one. In other words, an update is just unlink+link.

How is this implemented? For each `PrefixSetup` object passed to `UnlinkLinkTransaction`, a number of `ActionGroup` namedtuples (one per task *category*) will be instantiated and grouped together in a `PrefixActions`. These are then passed to `.verify()`. This method will take each action, run its checks and, if all of them passed, will allow us to perform the actual execution in `.execute()`. If one of them fails, the transaction can be aborted and rolled back.

For all this to work, each action object follows the `PathAction` API contract:

```
class PathAction:
    _verified = False

    def verify(self):
        "Run checks to assess if the action can proceed"

    def execute(self):
        "Perform the action"

    def reverse(self):
        "Undo execute"

    def cleanup(self):
        "Remove artifacts from verification, execution or reversal"

    @property
    def verified(self):
        "True if verification was run and successful"
```

Additional `PathAction` subclasses will add more methods and properties, but this is what the transaction execution logic expects. To support all the different actions involved in populating the prefix, the `PathAction` class tree holds quite the graph:

```
PathAction
  PrefixPathAction
    CreateInPrefixPathAction
      LinkPathAction
        PrefixReplaceLinkAction
      MakeMenuAction
      CreatePythonEntryPointAction
      CreatePrefixRecordAction
      UpdateHistoryAction
    RemoveFromPrefixPathAction
      UnlinkPathAction
        RemoveLinkedPackageRecordAction
      RemoveMenuAction
  RegisterEnvironmentLocationAction
  UnregisterEnvironmentLocationAction
  CacheUrlAction
  ExtractPackageAction

MultiPathAction
  CompileMultiPycAction
    AggregateCompileMultiPycAction
```

You are welcome to read on the docstring for each of those classes to understand which each one is doing; all of them are listed under `conda.core.path_actions`. In the following sections, we will only comment on the most important ones.

### Linking the files in the environment

When conda *links* a file from the cache location to the prefix location, it can actually mean three different actions:

1. Creating a soft link
2. Creating a hard link
3. Copying the file

The difference between soft links and hard links is subtle, but important. You can find more info on the differences elsewhere (e.g. here), but for our purposes it means that:

- Hard links are cheaper to resolve, behave like a real file, but can only link files in the same mount point.
- Soft links can link files across mount points, but they don't behave exactly like files (more like forwarders), so it's possible that they break assumptions made in certain pieces of code.

Most of the time, `conda` will try to hard link files and, if that fails, it will copy them over. Copying a file is an expensive disk operation, both in terms of time and space, so it should be the last option. However, sometimes it's the only way. Especially, when the file needs to be modified to be used in the target prefix.

Ummm… what? Why would `conda` modify a file to install it? This has to do with relocatability. When a `conda` package is created, `conda-build` creates up to three temporary environments:

- Build environment: where compilers and other build tools are installed, separate from the host environment to support cross-compilation.

- Host environment: where build-time dependencies are installed, together with the package you are building.
- Test environment: where run-time dependencies are installed, together with the package you just built. It simulates what will happen when a user installs the package so you can run arbitrary checks on your package.

When you are building a package, references to the build-time paths can leak into the content of some files, both text and binary. This is not a problem for users who build their own packages from source, since they can choose this path and leave the files there. However, this is almost never true for `conda` packages. They are created in one machine and installed in another. To avoid "path not found" issues and other problems, `conda-build` marks those packages that hold references to the build-time paths by replacing them with placeholders. At install-time, `conda` will replace those placeholders with the target prefix and everything works!

But there's a problem: we can't modify the files on the cache location because they might be used across environments (with obviously different paths). In these cases, files are not linked, but copied; the path replacement only happens on the target copy, of course!

How does `conda` know how to link a given package or, more precisely, its extracted files? All of this is determined in the preparation routines contained in `UnlinkLinkTransaction._prepare()` (more specifically, through `determine_link_type()`), as well as `LinkPathAction.create_file_link_actions()`.

Note that the (un)linking actions also include the execution of pre-(un)link and post-(un)link scripts, if listed.

### Action groups and actions, in detail

Once the old packages have been removed and the new ones have been linked through the appropriate means, we are done, right? Not yet! There's one step left: the post-linking logic.

It turns out that there's a number of smaller tasks that need to happen to make `conda` as convenient as it is. You can find all of them listed a few paragraphs above, but we'll cover them here, too. The execution order is determined in `UnlinLinkTransaction._execute`. All the possible groups are listed under `PrefixActions`. Their order is roughly how they happen in practice:

1. `remove_menu_action_groups`, composed of `RemoveMenuAction` actions.
2. `unlink_action_groups`, includes `UnlinkPathAction`, `RemoveLinkedPackageRecordAction`, as well as the logic to run the pre- and post-unlink scripts.
3. `unregister_action_groups`, basically a single `UnregisterEnvironmentLocationAction` action.
4. `link_action_groups`, includes `LinkPathAction`, `PrefixReplaceLinkAction`, as well as the logic to run pre- and post-link scripts.
5. `entry_point_action_groups`, a collection of `CreatePythonEntryPointAction` actions.
6. `register_action_groups`, a single `RegisterEnvironmentLocationAction` action.
7. `compile_action_groups`, several `CompileMultiPycAction` that end up aggregated as a `AggregateCompileMultiPycAction` for performance.
8. `make_menu_action_groups`, composed of `MakeMenuAction` actions.
9. `prefix_record_groups`, records installed packages in the environment via `CreatePrefixRecordAction` actions.
10. `initial_action_groups`, includes any plugin-defined pre-transaction actions.
11. `final_action_groups`, includes any plugin-defined post-transaction actions.

Let's discuss these actions groups for the command we are describing in this guide: `conda install numpy`. The solution given by the solver says we need to:

- unlink Python 3.9.6

- link Python 3.9.9
- link numpy 1.19

This is what would happen:

1. No menu items are removed because Python 3.9.6 didn't create any.
2. Pre-unlink scripts for Python 3.9.6 would run, but in this case there are none.
3. Python 3.9.6 files are removed from the environment. This can be parallelized.
4. Post-unlink scripts are run, if any.
5. Pre-link scripts are run for Python 3.9.9 and numpy 1.19, if any.
6. Files in the Python 3.9.9 and numpy 1.19 packages are linked and/or copied to the prefix. This can be parallelized.
7. Entry points are created for the new packages, if any.
8. Post-link scripts are run.
9. `pyc` files are generated for the new packages.
10. The new packages are registered under `conda-meta/`.
11. The menu shortcuts are created for the new packages, if any.

Any of these steps can fail with a given exception. If that's the case, the first of those exceptions is printed to STDOUT. Additionally, if `rollback_enabled` is properly configured in the `context`, the transaction will be rolled back by calling the `.reverse()` method in each action, from last to first.

If no exceptions are reported, then the actions can run their cleanup routines.

And that's it! If this command had resulted in a new environment being created, you would get a message telling you how to activate the newly created environment.

### Conclusion

This is what happens when you type `conda install`. It might be a bit more involved than you initially thought, but it all boils down to only some steps. TL;DR:

1. Parse arguments and initialize the context
2. Download and build the index
3. Tell the solver what we want
4. Convert the solution into a transaction
5. Verify and run each action contained in the transaction

### `conda init` and `conda activate`

`conda` ships *virtual environments* by design. When you install Anaconda or Miniconda, you obtain a `base` environment that is essentially a regular environment with some extra checks. These checks have to do with what the `conda` command really is and how it is installed in your system.

**Base prefix vs target prefix**

Originally, the base installation for `conda` was called the *root* environment. Every other environment lived under `envs/` in that root environment. The root environment was later renamed to *base*, but the code still distinguishes between base and target using the old terminology:

- `context.root_prefix`: the path where the `base` conda installation is located.
- `context.target_prefix`: the environment `conda` is running a command on. Usually defaults to the activated environment, unless `-n` (name) or `-p` (prefix) is specified in the command line. Note that if you are operating on the `base` environment, the target prefix will have the same value as the root prefix.

When you type `conda` in your terminal, your shell will try to find either:

- a shell function named `conda`
- an executable file named `conda` in your `PATH` directories

If your `conda` installation has been properly initialized, it will find the shell function. If not, it might find the `conda` executable if it *happens* to be in `PATH`, but this is most often not the case. That's why initialization is there to begin with!

### Conda initialization

Why is initialization needed at all to begin with? There are several reasons:

- Activation requires interacting with the shell context very closely
- It does not pollute `PATH` unnecessarily
- Improves performance in certain operations

The main idea behind initialization is to provide a `conda` shell function that allows the Python code to interact with the shell context more intimately. It also allows a cleaner `PATH` manipulation and snappier responses in some `conda` commands.

The `conda` shell function is mainly a forwarder function. It will delegate most of the commands to the real `conda` executable driven by the Python library. However, it will intercept two very specific subcommands:

- `conda activate`
- `conda deactivate`

This interception is needed because activation/deactivation requires exporting (or unsetting) environment variables back to the shell session (and not just temporarily in the Python process). This will be discussed in the next section.

So how is initialization performed? This is the job of the `conda init` subcommand, driven by the `conda.cli.main_init` module, which depends direcly on the `conda.core.initialize` module. Let's see how this is implemented.

`conda init` will initialize a shell permanently by writing some shell code in the relevant startup scripts of your shell (e.g. `~/.bashrc`). This is done through different functions defined in `conda.core.initialize`, namely:

- `init_sh_user`: initializes a Posix shell (like Bash) for the current user.
- `init_sh_system`: initializes a Posix shell (like Bash) globally, for all users.
- `init_fish_user`: initializes the Fish shell for the current user.
- `init_xonsh_user`: initializes the Xonsh shell for the current user.

- `init_cmd_exe_registry`: initializes Cmd.exe through the Windows Registry.
- `init_powershell_user`: initializes Powershell for the current user.
- `init_long_path`: configures Windows to support longer paths.

What each function does depends on the nature of each shell. In the case of Bash shells, the underlying `Activator` subclass (more below) can generate the hook code dynamically. In other Posix shells and Powershell, a script is sourced from its location in the `base` environment. With Cmd, the changes are introduced through the Windows Registry. The end result is the same: they will end up defining a `conda` shell function with the behavior described above.

### Conda activate

All `Activator` classes can be found under `conda.activate`. Their job is essentially to write shell-native code programmatically. As of conda 4.11, these are the supported shells and their corresponding activators

- `posix`, `ash`, `bash`, `dash`, `zsh`: all driven by `PosixActivator`.
- `csh`, `tcsh`: `CshActivator`.
- `xonsh`: `XonshActivator`.
- `cmd.exe`: `CmdExeActivator`.
- `fish`: `FishActivator`.
- `powershell`: `PowerShellActivator`.

You can add all these classes through the `conda shell.<key>` command, where `key` is any of the names in the list above. These CLI interface offers several subcommands, connected directly to methods of the same name:

- `activate`: writes the shell code to activate a given environment.
- `deactivate`: writes the shell code to deactivate a given environment.
- `hook`: writes the shell code to register the initialization code for the `conda` shell code.
- `commands`: writes the shell code needed for autocompletion engines.
- `reactivate`: writes the shell code for deactivation followed by activation.

To be clear, we are saying these functions only *write* shell code. They do *not* execute it! This needs to be done by the shell itself! That's why we need a `conda` shell function, so these shell strings can be `eval`'d or `source`'d in-session.

Let's see what happens when you run `conda shell.bash activate`:

```
$ conda shell.bash activate
export PATH='/Users/username/.local/anaconda/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/
↪usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/anaconda/condabin:/
↪opt/homebrew/bin:/opt/homebrew/sbin'
unset CONDA_PREFIX_1
PS1='(base) '
export CONDA_PREFIX='/Users/username/.local/anaconda'
export CONDA_SHLVL='1'
export CONDA_DEFAULT_ENV='base'
export CONDA_PROMPT_MODIFIER='(base) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
```

See? It only wrote some shell code to stdout, but it wasn't executed. We would need to do this to actually run it:

```
$ eval "$(conda shell.bash activate)"
```

And this is essentially what `conda activate` does: it calls the registered shell activator to obtain the required shell code and then it `eval`s it. In some shells with no `eval` equivalent, a temporary script is written and sourced or called. The final effect is the same.

Ok, but what is that shell code doing? Mainly setting your `PATH` correctly so the executables of your `base` environment can be found (like `python`). It also sets some extra variables to keep a reference to the path of the currently active environment, the shell prompt modifiers and other information for `conda` internals.

This command can also generate the code for any other environment you want, not just `base`. Just pass the name or path:

```
$ conda shell.bash activate mamba-poc
PS1='(mamba-poc) '
export PATH='/Users/username/.local/anaconda/envs/mamba-poc/bin:/opt/homebrew/bin:/opt/
↪homebrew/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/
↪anaconda/condabin:/opt/homebrew/bin:/opt/homebrew/sbin'
export CONDA_PREFIX='/Users/username/.local/anaconda/envs/mamba-poc'
export CONDA_SHLVL='2'
export CONDA_DEFAULT_ENV='mamba-poc'
export CONDA_PROMPT_MODIFIER='(mamba-poc) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
export CONDA_PREFIX_1='/Users/username/.local/anaconda'
```

Now the paths are different, as well as some numbers (e.g. `CONDA_SHLVL`). This is used by conda to keep track of what was activated before, so when you deactivate the last one, you can get back to the previous one seamlessly.

### Activation/deactivation scripts

The activation/deactivation code can also include calls to activation/deactivation scripts. If present in the appropriate directories for your shell (e.g. `CONDA_PREFIX/etc/conda/activate.d/`), they will be called before deactivation or after activation, respectively. For example, compilers usually set up some environment variables to help configure the default flags. This is what happens when you activate an environment that contains Clang and Gfortran:

```
$ conda shell.bash activate compilers
PS1='(compilers) '
export PATH='/Users/username/.local/anaconda/envs/compilers/bin:/opt/homebrew/bin:/opt/
↪homebrew/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/
↪anaconda/condabin:/opt/homebrew/bin:/opt/homebrew/sbin'
export CONDA_PREFIX='/Users/username/.local/anaconda/envs/compilers'
export CONDA_SHLVL='2'
export CONDA_DEFAULT_ENV='compilers'
export CONDA_PROMPT_MODIFIER='(compilers) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
export CONDA_PREFIX_1='/Users/username/.local/anaconda'
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate-gfortran_
```

(continues on next page)

(continued from previous page)

```
↪osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate_clang_
↪osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/activate.d/activate_clangxx_
↪osx-arm64.sh"
```

Those three lines are sourcing the relevant scripts. Similarly, for deactivation, notice how the deactivation scripts are executed first this time:

```
$ conda shell.bash deactivate
export PATH='/Users/username/.local/anaconda/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/
↪usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/username/.local/anaconda/condabin:/
↪opt/homebrew/bin:/opt/homebrew/sbin'
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate_
↪clangxx_osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate_
↪clang_osx-arm64.sh"
. "/Users/username/.local/anaconda/envs/compilers/etc/conda/deactivate.d/deactivate-
↪gfortran_osx-arm64.sh"
unset CONDA_PREFIX_1
PS1='(base) '
export CONDA_PREFIX='/Users/username/.local/anaconda'
export CONDA_SHLVL='1'
export CONDA_DEFAULT_ENV='base'
export CONDA_PROMPT_MODIFIER='(base) '
export CONDA_EXE='/Users/username/.local/anaconda/bin/conda'
export _CE_M=''
export _CE_CONDA=''
export CONDA_PYTHON_EXE='/Users/username/.local/anaconda/bin/python'
```

### `conda config` and context

The `context` object is central to many parts of the `conda` codebase. It serves as a centralized repository of settings. You normally import the singleton and access its (many) attributes directly:

```
from conda.base.context import context

context.quiet
# False
```

This singleton is initialized from a cascade of different possible sources. From lower to higher precedence:

1. Default values hardcoded in the `Context` class. These are defined via class attributes.
2. Values defined in the configuration files (`.condarc`), which have their own *precedence*.
3. Values set by the corresponding command line arguments, if any.
4. Values defined by their corresponding `CONDA_*` environment variables, if present.

The mechanism implementing this behavior is an elaborate object with several types of objects involved.

### Anatomy of the `Context` class

`conda.base.context.Context` is an conda-specific subclass of the application-agnostic `conda.common.configuration.Configuration` class. This class implements the precedence order for the instantiation of each defined attribute, as well as the overall validation logic and help message reporting. But that's it, it's merely a storage of `ParameterLoader` objects which, in turn, instantiate the relevant `Parameter` subclasses in each attribute. Roughly:

```
class MyConfiguration(Configuration):
    string_field = ParameterLoader(PrimitiveParameter("default", str))
    list_of_int_field = ParameterLoader(SequenceParameter([1, 2, 3], int))
    map_of_foat_values_field = ParameterLoader(MapParameter({"key": 1.0}, float))
```

When `MyConfiguration` is instantiated, those class attributes are populated by the `.raw_data` dicionary that has been filled in with the values coming from the precedence chain stated above. The `raw_data` dictionary contains `RawParameter` objects, subclassed to deal with the specifics of their origin (YAML file, environment variable, command line flag). Each `ParameterLoader` object will pass the `RawParameter` object to the `.load()` method of its relevant `Parameter` subclass, which are designed to return their corresponding `LoadedParameter` object counterpart.

It's a bit confusing, but the delegation happens like this:

1. The `Configuration` subclass parses the raw values of the possible origins and stores them as the relevant `RawParameter` objects, which can be:
    - `EnvRawParameter`: for those coming from an environment variable
    - `ArgParseRawParameter`: for those coming from a command line flag
    - `YamlRawParameter`: for those coming from a configuration file
    - `DefaultValueRawParameter`: for those coming from the default value given to `ParameterLoader`
2. Each `Configuration` attribute is a `ParameterLoader`, which implements the `property` protocol via `__get__`. This means that, upon attribute access (e.g. `MyConfiguration.string_field`), the `ParameterLoader` can execute the loading logic. This means finding potential type matches in the raw data, loading them as `LoadedParameter` objects and merging them with the adequate precedence order.

The merging policy depends on the `(Loaded)Parameter` subtype. Below is a list of available subtypes:

- `PrimitiveParameter`: holds a single scalar value of type `str`, `int`, `float`, `complex`, `bool` or `NoneType`.
- `SequenceParameter`: holds an iterable (`list`) of other `Parameter` objects.
- `MapParameter`: holds a mapping (`dict`) of other `Parameter` objects.
- `ObjectParameter`: holds an object with attributes set to `Parameter` objects.

The main goal of the `Parameter` objects is to implement how to typify and turn the raw values into their `Loaded` counterparts. These implement the validation routines and define how parameters for the same key should be merged:

- `PrimitiveLoadedParameter`: value with highest precedence replaces the existing one.
- `SequenceLoadedParameter`: extends with no duplication, keeping precedence.
- `MapLoadedParameter`: cascading updates, highest precedence kept.
- `ObjectLoadedParameter`: same as `Map`.

After all of this, the `LoadedParameter` objects are *typified*: this is when type validation is performed. If everything goes well, you obtain your values just fine. If not, the validation errors are raised.

Take into account that the result is cached for faster subsequent access. This means that even if you change the value of the environment variables responsible for a given setting, this won't be reflected in the `context` object until you refresh it with `conda.base.context.reset_context()`.

**Do not modify the Context object!**

`ParameterLoader` does not implement the `__set__` method of the `property` protocol, so you can freely override an attribute defined in a `Configuration` subclass. You might think that this will redefine the value after passing through the validation machinery, but that's not true. You will simply overwrite it entirely with the raw value and that's probably not what you want.

Instead, consider the `context` object immutable. If you need to change a setting at runtime, it is probably *A Bad Idea*. The only situation where this is acceptable is during testing.

### Setting values in the different origins

There's some magic behind the possible origins for the settings values. How these are tied to the final `Configuration` object might not be obvious at first. This is different for each `RawParameter` subclass:

- `DefaultValueRawParameter`: Users will never see this one. It only wraps the default value passed to the `ParameterLoader` class. Safe to ignore.
- `YamlRawParameter`: This one takes a YAML file and parses it as a dictionary. The keys in this file must match the attribute names in the `Configuration` class exactly (or one of their aliases). Matching happens automatically once this is properly set up. How the values are parsed depends on the YAML Loader, set internally by `conda`.
- `EnvRawParameter`: Values coming from certain environment variables can make it to the `Configuration` instance, provided they are formatted as `<APP_NAME>_<PARAMETER_NAME>`, all uppercase. The app name is defined by the `Configuration` subclass. The parameter name is defined by the attribute name in the class, transformed to upper case. For example, `context.ignore_pinned` can be set with `CONDA_IGNORE_PINNED`. The value of the variable is parsed in different ways depending on the type:
  - `PrimitiveParameter` is the easy one. The environment variable string is parsed as the expected type. Booleans are a bit different since several strings are recognized as such, and in a case-insensitive way:
    - `True` can be set with `true`, `yes`, `on` and `y`.
    - `False` can be set with `false`, `off`, `n`, `no`, `non`, `none` and `""` (empty string).
  - `SequenceParameter` can specify their own delimiter (e.g. `,`), so the environment variable string is processed into a list.
  - `MapParameter` and `ObjectParameter` do not support being set with environment variables.
- `ArgParseRawParameter`: These are a bit different because there is no automated mechanism that ties a given command line flag to the context object. This means that if you add a new setting to the `Context` class and you want that available in the CLI as a command line flag, you have to add it yourself. If that's the case, refer to `conda.cli.conda_argparse` and make sure that the `dest` value of your `argparse.Argument` matches the attribute name in `Context`. This way, `Configuration.__init__` can take the `argparse.Namespace` object, turn it into a dictionary, and make it pass through the loading machinery.

### Solvers

The guide *conda install* didn't go into details of the solver black box. It did mention the high-level `Solver` API and how `conda` expects a transaction out of it, but we never got to learn what happens *inside* the solver itself. We only covered these three steps:

> The details are complicated, but in essence, the solver will:
>
> 1. Express the requested packages, command line options and prefix state as `MatchSpec` objects
> 2. Query the index for the best possible match that satisfy those constraints
> 3. Return a list of `PackageRecord` objects.

How do we transform the prefix state and configurations into a list of `MatchSpec` objects? How are those turned into a list of `PackageRecord` objects? Where are those `PackageRecord` objects coming from? We are going to cover these aspects in detail here.

#### `MatchSpec` vs `PackageRecord`

First, let's define what each object does:

- `PackageRecord` objects represent a concrete package tarball and its contents. They follow specific naming conventions and expose several fields. Inspect them directly in the class code.
- `MatchSpec` objects are essentially a query language to find `PackageRecord` objects. Internally, `conda` will translate your command line requests, like `numpy>=1.19`, `python=3.*` or `pytorch=1.8.*=*cuda*`, into instances of this class. This query language has its own syntax and rules, detailed here. The most important fields of a `MatchSpec` object are:
  - `name`: the name of the package (e.g. `pytorch`); always expected.
  - `version`: the version constraints (e.g. `1.8.*`); can be empty but if `build` is set, set it to `*` to avoid issues with the `.conda_build_form()` method.
  - `build`: the build string constraints (e.g. `*cuda*`); can be empty.

**Create a MatchSpec object from a PackageRecord instance**

You can create a `MatchSpec` object from a `PackageRecord` instance using the `.to_match_spec()` method. This will create a `MatchSpec` object with its fields set to exactly match the originating `PackageRecord`.

Note that there are two `PackageRecord` subclasses with extra fields, so we need to distinguish between three types, all of them useful:

- `PackageRecord`: A record as present in the index (channel).
- `PackageCacheRecord`: A record already extracted in the cache. Contains extra fields for the tarball path in disk and its extracted directory.
- `PrefixRecord`: A record installed in a prefix. Same as above, plus fields for the files that make the package and how they were linked in the prefix. It can also host information about which `MatchSpec` string resulted in this record being installed.

### Remote state: the index

So the solver takes `MatchSpec` objects, queries the index for the best match and returns `PackageRecord` objects. Perfect. What's the index? It's the result of aggregating the requested conda channels in a single entity. For more information, check *Fetching the index*.

### Local state: the prefix and context

When you do `conda install numpy`, do you think the solver will just see something like `specs=[MatchSpec("numpy")]`? Well, not that quick. The explicit instructions given by the user are only one part of the request we will send to the solver. Other pieces of implicit state are taken into account to build the final request. Namely, the state of your prefix. In total, these are the ingredients of the solver request:

1. Packages already present in your environment, if you are not *creating* a new one. This is exposed through the `conda.core.prefix_data.PrefixData` class, which provides an iterator method via `.iter_records()`. As we saw before, this yields `conda.models.records.PrefixRecord` objects, a `PackageRecord` subclass for installed records.
2. Past actions you have performed in that environment; the *History*. This is a journal of all the `conda install|update|remove` commands you have run in the past. In other words, the *specs* matched by those previous actions will receive extra protections in the solver.
3. Packages included in the *aggressive updates* list. These packages are always included in any requests to make sure they stay up-to-date under all circumstances.
4. Packages pinned to a specific version, either via `pinned_packages` in your `.condarc` or defined in a `$PREFIX/conda-meta/pinned` file.
5. In new environments, packages included in the `create_default_packages` list. These specs are injected in each `conda create` command, so the solver will see them as explicitly requested by the user.
6. And, finally, the specs the user is asking for. Sometimes this is explicit (e.g. `conda install numpy`) and sometimes a bit implicit (e.g. `conda update --all` is telling the solver to add all installed packages to the update list).

All of those sources of information produce a number a of `MatchSpec` objects, which are then combined and modified in very specific ways depending on the command line flags and their origin (e.g. specs coming from the pinned packages won't be modified, unless the user asks for it explicitly). This logic is intricate and will be covered in the next sections. A more technical description is also available in *Technical specification: solver state*.

### The high-level logic in `conda.cli.install`

The full solver logic does not start at the `conda.core.solve.Solver` API, but before that, all the way up in the `conda.cli.install` module. Here, some important decisions are already made:

- Whether the solver is not needed at all because:
  - The operation is an explicit package install
  - The user requested to roll back to a history checkpoint
  - We are just creating a copy of an existing environment (cloning)
- Which `repodata` source to use (see *here*). It not only depends on the current configuration (via `.condarc` or command line flags), but also on the value of `use_only_tar_bz2`.
- Whether the solver should start by freezing all installed packages (default for `conda install` and `conda remove` in existing environments).

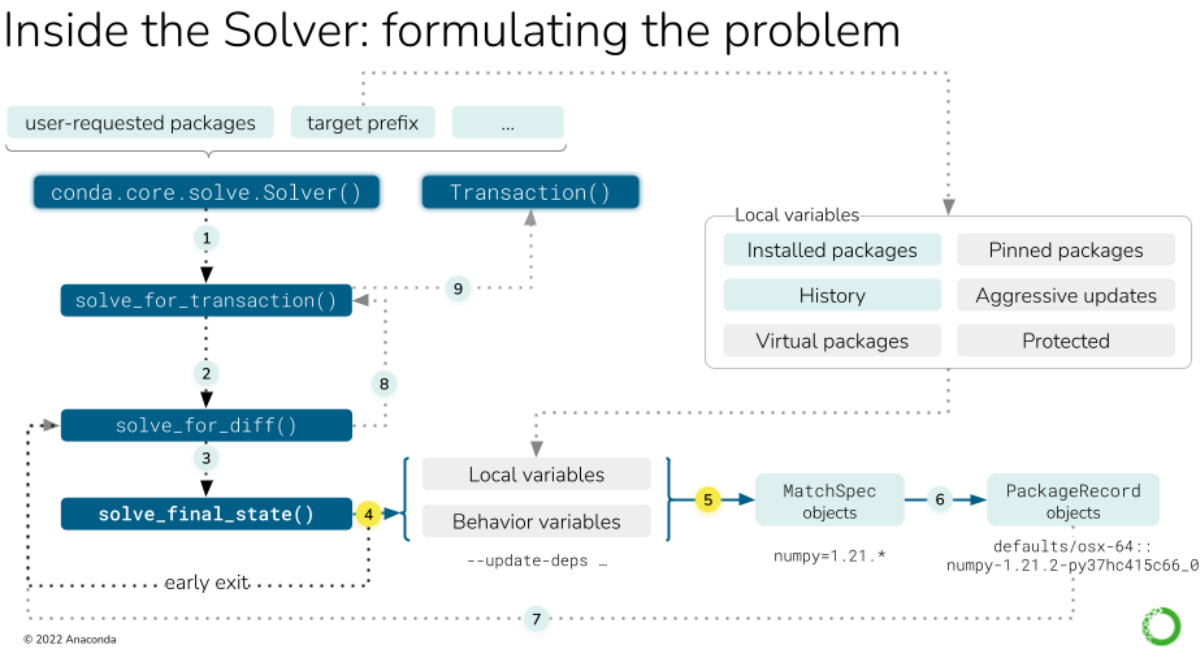



Fig. 2: Local variables affect the solving process explicitly and implicitly. As seen in *in the conda install deep dive*, the main actor is the `conda.core.solve.Solver` class. Before invoking the SAT solver, we can describe nine steps:

1. Instantiate the `Solver` class with the user-requested packages and the active environment (target prefix)
2. Call the `solve_for_transaction()` method on the instance, which calls `solve_for_diff()`.
3. Call `solve_final_state()`, which takes some more arguments from the CLI.
4. Under some circumstances, we can return early (e.g. the packages are already installed).
5. If we didn't return early, we collect all the local variables into a list of `MatchSpec` objects.

For steps six to nine, see *this figure*.

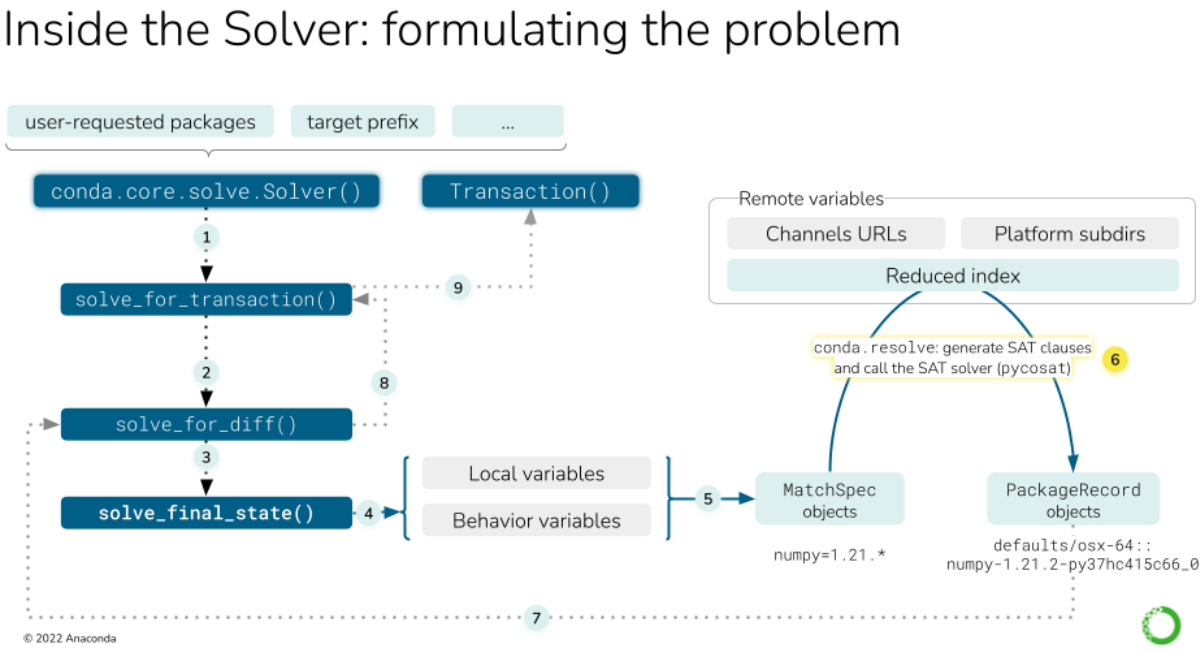



Fig. 3: The remote variables in a solve refer to, essentially, the package index (channels). This figure describes nine steps, focusing on 6-9. For steps 1-5, see *the previous figure*.

6. All the channels need to be fetched by now, but they have to be aggregated and reduced so the solver only handles the relevant bits. This step transforms "channels" into a list of available `PackageRecord` objects.
7. This is where the SAT solver will act. It will use the list of `MatchSpec` objects to pick a number of `PackageRecord` entries from the index, thus building the "final state of the solved environment". This is detailed later in this deep dive guide, if you need more info.
8. `solve_for_diff` takes the final state and compares it to the initial state, generating the differences between them (e.g. package A was updated to version 1.2, package B was removed).
9. `solve_for_transaction` takes the diff and some more metadata in the instance to generate the `Transaction` object.

- If the solver does not find a solution, whether we need to retry again without freezing the installed packages for the current `repodata` variant or if we should try with the next one.

So, roughly, the global logic there follows this pseudocode:

```
if operation in (explicit, rollback, clone):
    transaction = handle_without_solver()
else:
    repodatas = from_config or ("current_repodata.json", "repodata.json")
    freeze = (is_install or is_remove) and env_exists and update_modifier not in argv
    for repodata in repodatas:
        try:
            transaction = solve_for_transaction(...)
        except:
            if repodata is last:
                raise
            elif freeze:
                transaction = solve_for_transaction(freeze_installed=False)
            else:
                continue  # try next repodata

handle_txn(transaction)
```

Check *this other figure* for a schematic representation of this pseudocode.

We have, then, two reasons to re-run the full solver logic:

- Freezing the installed packages didn't work, so we try without freezing again.
- Using `current_repodata` did not work, so we try with full `repodata`.

These two strategies are stacked so in the end, before eventually failing, we will have tried four things:

1. Solve with `current_repodata.json` and `freeze_installed=True`
2. Solve with `current_repodata.json` and `freeze_installed=False`
3. Solve with `repodata.json` and `freeze_installed=True`
4. Solve with `repodata.json` and `freeze_installed=False`

Interestingly, those strategies are designed to improve `conda`'s average performance, but they should be seen as a risky bet. Those attempts can get expensive!

**How to ask for a simpler approach**

If you want to try the full thing without checking whether the optimized solves work, you can override the default behaviour with these flags in your `conda install` commands:

- `--repodata-fn=repodata.json`: do not use `current_repodata.json`
- `--update-specs`: do not try to freeze installed

Then, the `Solver` class has its own internal logic, which also features some retry loops. This will be discussed later and summarized.

### Early exit tasks

Some tasks do not involve the solver at all. Let's enumerate them:

- Explicit package installs: no index or prefix state needed.
- Cloning an environment: the index might be needed if the cache has been cleared.
- History rollback: currently broken.
- Forced removal: prefix state needed. This happens in the `Solver` class.
- Skip solve if already satisfied: prefix state needed. This happens in the `Solver` class.

### Explicit package installs

These commands do not need a solver because the requested packages are expressed with a direct URL or path to a specific tarball. Instead of a `MatchSpec`, we already have a `PackageRecord`-like entity! For this to work, all the requested packages neeed to be URLs or paths. They can be typed in the command line or in a text file including a `@EXPLICIT` line.

Since the solver is not involved, the dependencies of the explicit package(s) are not processed at all. This can leave the environment in an *inconsistent state*, which can be fixed by running `conda update --all`, for example.

Explicit installs are taken care of by the `explicit` function.

### Cloning an environment

`conda create` has a `--clone` flag that allows you to create a fully-working copy of an existing environment. This is needed because you cannot relocate an environment using `cp`, `mv`, or your favorite file manager without unintended consequences. Some files in a conda environment might contain hardcoded paths to existing files in the original location, and those references will break if `cp` or `mv` is utilized (conda environments *can* be renamed via the `conda rename` command, however; see the *following section* for more information).

The `clone_env` function implements this functionality. It essentially takes the source environment, generates the URLs for each installed packages (filtering `conda`, `conda-env` and their dependencies) and passes the list of URLs to `explicit()`. If the source tarballs are not in the cache anymore, it will query the index for the best possible match for the current channels. As such, there's a slim chance that the copy is not exactly a clone of the original environment.

### Renaming an environment

When the `conda rename` command is used to rename an already-existing environment, please keep in mind that the solver is not invoked at all, since the command essentially does a `conda create --clone` and `conda remove --all` of the environment.

### History rollback

`conda install` has a `--revision` flag, which allows you to revert the state of the environment to a previous one. This is done through the `History` file, but its current implementation can be considered broken. Once fixed, we will cover it in detail.

### Forced removals

Similar to explicit installs, you can remove a package without performing a full solve. If `conda remove` is invoked with `--force`, the specified package(s) will be removed directly, without analyzing their dependency tree and pruning the orphans. This can only happen after querying the active prefix for the installed packages, so it is handled in the `Solver` class. This part of the logic returns the list of `PackageRecord` objects already found in the `PrefixData` list after filtering out the ones that should be removed.

### Skip solve if already satisfied

`conda install` and `update` have a rather obscure flag: `-S, --satisfied-skip-solve`:

> Exit early and do not run the solver if the requested specs are satisfied. Also skips aggressive updates as configured by 'aggressive_update_packages'. Similar to the default behavior of 'pip install'.

This is also implemented at the `Solver` level, because we also need a `PrefixData` instance. It essentially checks if all of the passed `MatchSpec` objects can match a `PackageRecord` already in prefix. If that's the case, we return the installed state as-is. If not, we proceed for the full solve.

### Details of `Solver.solve_final_state()`

> **Note**
>
> From here on, the document only covers the classic solver logic (which uses `pycosat`). The `libmamba` solver has a different approach and is not documented here. Please refer to its documentation for more information.

This is where most of the intricacies of the `conda` logic are defined. In this step, the configuration, command line flags, user-requested specs and prefix state are aggregated to query the current index for the best match.

The aggregation of all those state bits will result in a list of `MatchSpec` objects. While it's easy to establish which package names will make it to the list, deciding which version and build string constraints the specs carry is a bit more involved.

This is currently implemented in the `conda.core.solve.Solver` class. Its main goal is to populate the `specs_map` dictionary, which maps package names (`str`) to `MatchSpec` objects. This happens at the beginning of the `.solve_final_state()` method. The full details of the `specs_map` population are covered in the *solver state technical specification*, but here's a little map of what submethods are involved:

1. Initialization of the `SolverStateContainer`: Often abbreviated as `ssc`, it's a helper class to store some state across attempts (remember there are several retry loops). Most importantly, it stores two key attributes (among others):
   - `specs_map`: same as above. This is where it lives across solver attempts.
   - `solution_precs`: a list of `PackageRecord` objects. It stores the solution returned by the SAT solver. It's always initialized to reflect the installed packages in the target prefix.

2. `Solver._collect_all_metadata()`: Initializes the `specs_map` with the specs found in the history or with the specs corresponding to the installed records. This method delegates to `Solver._prepare()`. This initializes the index by fetching the channels and reducing it. Then, a `conda.resolve.Resolve` instance is created with that index. The index is stored in the `Solver` instance as `._index` and the `Resolve` object as `._r`. They are also kept around in the `SolverStateContainer`, but as public attributes: `.index` and `.r`, respectively.
3. `Solver._remove_specs()`: If `conda remove` was called, it removes the relevant specs from `specs_map`.
4. `Solver._add_specs()`: For all the other `conda` commands (`create`, `install`, `update`), it adds (or modifies) the relevant specs to `specs_map`. This is one of the most complicated pieces of logic in the class!

**Check the other parts of the Solver API**

You can check the rest of the Solver API *here*.

At this point, the `specs_map` is adequately populated and we can call the SAT solver wrapped by the `conda.resolve.Resolve` class. This is done in `Solver._run_sat()`, but this method does some other things before actually solving the SAT problem:

- Before calling `._run_sat()`, inconsistency analysis is performed via `Solver._find_inconsistent_packages`. This will preemptively remove certain `PackageRecord` objects from `ssc.solution_precs` if `Resolve.bad_installed()` determined they were causing inconsistencies. This actually runs a series of small solves to check that the installed records form a satisfiable set of clauses. Those that prevent that solution from being found are annotated as such and ignored during the real solve later.
- Make sure the requested package names are available in the index.
- Anticipate and minimize potentially conflicting specs. This happens in a `while` loop fed by `Resolve.get_conflicting_specs()`. If a spec is found to be conflicting, it is *neutered*: a new `MatchSpec` object is created, but without version and build string constrains (e.g. `numpy >=1.19` becomes just `numpy`). Then, `Resolve.get_conflicting_specs()` is called again, and the loop continues until convergence: the list of conflicts cannot be reduced further, either because there are no conflicts left or because the existing conflicts cannot be resolved by constraint relaxation.
- Now, the SAT solver is called. This happens via `Resolve.solve()`. More on this below.
- If the solver failed, then `UnsatisfiableError` is raised. Depending on which attempt we are on, `conda` will try again with non-frozen installed packages or a different repodata, or it will give up and analyze the conflict cause core. This will be detailed later.
- If the solver succeeded, some bookkeeping needs to be done:
  - Neutered specs that happened to be in the history are annotated as such.
  - Inconsistent packages are added back to the solution, including potential orphans.
  - Constraint analysis is run via `Solver.get_constrained_packages()` and `Solver.determine_constricting_specs()` to help the user understand why some packages were not updated.

We are not done yet, though. After `Solver._run_sat()`, we still need to run the post-solver logic! After the solve, the final list of `PackageRecord` objects might still change if certain modifiers are set. This is handled in the `Solver._post_sat_handling()`:

- `--no-deps` (`DepsModifier.NO_DEPS`): Remove dependencies of the explicitly requested packages from the final solution.
- `--only-deps` (`DepsModifier.ONLY_DEPS`): Remove explicitly requested packages from the final solution but leave their dependencies. This is done via `PrefixGraph.remove_youngest_descendant_nodes_with_specs()`.

- `--update-deps` (`UpdateModifier.UPDATE_DEPS`): This is the most interesting one. It actually runs a second solve (!) where the user-requested specs are the originally requested specs plus their (now determined) dependencies.
- `--prune`: Removes orphan packages from the solution.

**The Solver also checks for Conda updates**

Interestingly, the Solver API is also responsible of checking if new `conda` versions are available in the configured channels. This is done here to take advantage of the fact that the index has been already built for the rest of the class.

### Details of `conda.resolve.Resolve`

This is the class that actually wraps the SAT solver. `conda.core.solve.Solver` is a higher level API that configures the solver *request* and prepares the transaction. The actual solution is computed in this other module we are discussing now.

The `Resolve` object will mostly receive two arguments:

- The fetched `index`, as processed by `conda.index.get_index()`.
- The configured `channels`, so *channel priority* can be sorted out.

It will also hold certain states:

- The `index` will be grouped by name under a `.groups` dictionary (`str`, `[PackageRecord]`). Each group is later sorted so newer packages are listed first, helping reduce the index better.
- Another dictionary of `PackageRecord` groups will be created, keyed by their `track_features` entries, under the `.trackers` attribute.
- Some other dictionaries are initialized as caches.

The main methods in this class are:

- `bad_installed()`: This method uses a series of small solves to check if the installed packages are in a consistent state. In other words, if all the `PackageRecord` entries were expressed as `MatchSpec` objects, would the environment be solvable?
- `get_reduced_index()`: This method takes a full index and trims out the parts that are not necessary for the current request, thus reducing the solution space and speeding up the solver.
- `gen_clauses()`: This instantiates and configures the `Clauses` object, which is the real SAT solver wrapper. More on this later.
- `solve()`: The main method in the `Resolve` class. It will be discussed in the next section.
- `find_conflicts()`: If the solver didn't succeed, this method performs a conflict analysis to find the most plausible explanation for the current conflicts. It essentially relies on `build_conflict_map()` to "find the common dependencies that might be the cause of conflicts". `conda` can spend a lot of time in this method.

**Disabling conflict analysis**

Conflict analysis can be disabled through the `context.unsatisfiable_hints` options, but unfortunately that gets in the way of conda's iterative logic. It will shortcut early in the chain of attempts and prevent the solver from trying less constrained specs. This is a part of the logic that should be improved.

### Resolve.solve()

As introduced above, this is the main method in the Resolve class. It will perform the following actions:

1. Reduce the index via get_reduced_index. If unsuccessful, try to detect if packages are missing or the wrong version was requested. We can raise early to trigger a new attempt in conda.cli.install (remember, unfrozen or next repodata) or, if it's the last attempt, we go straight to find_conflicts() to understand what's wrong.
2. Instantiate a new Resolve object with the reduced index to generate the Clauses object via gen_clauses(). This method relies on push_MatchSpec() to turn the MatchSpec object into an SAT clause inside the Clauses object (referred to as C).
3. Run Clauses.sat() to solve the SAT problem. If a solution cannot be found, deal with the error in the usual way: raise early to trigger another attempt or call find_conflicts() to try explaining why.
4. If no errors are found, then we have one or more solutions available, and we need to post-process them to find the *best* one. This is done in several steps:
    1. Minimize the amount of removed packages. The SAT clauses are generated via Resolve.generate_removal_count() and then Clauses.minimize() will use it to optimize the current solution.
    2. Maximize how well each record in the solution matches the spec. The SAT clauses are now generated in Resolve.generate_version_metrics(). This returns five sets of clauses: channel, version, build, arch or noarch, and timestamp. At this point, only channels and versions are optimized.
    3. Minimize the number of records with track_feature entries. SAT clauses are coming from Resolve.generate_feature_count().
    4. Minimize the number of records with features entries. SAT clauses are coming from Resolve.generate_feature_metric().
    5. Now, we continue the work started at (2). We will maximize the build number and choose arch-specific packages over noarch variants.
    6. We also want to include as many *optional* specs in the solution as possible. Optimize for that thanks to the clauses generated by Resolve.generate_install_count().
    7. At the same time, we will minimize the number of necessary updates if keeping the installed versions also satisfies the request. Clauses generated with Resolve.generate_update_count().
    8. Steps (2) and (5) are also applied to indirect dependencies.
    9. Minimize the number of packages in the solution. This is done by removing unnecessary packages.
    10. Finally, maximize timestamps until convergence so the most recent packages are preferred.
5. At this point, the SAT solution indices can be translated back to *SAT names*. This is done in the clean() local function you can find in Resolve.sat().
6. There's a chance we can find alternate solutions for the problem, and this is explored now, but eventually only the first one will be returned while translating the *SAT names* to PackageRecord objects.

### The `Clauses` object wraps the SAT solver using several layers

The `Resolve` class exposes the solving logic, but when it comes to interacting with the SAT solver engine, that's done through the `Clauses` object tree. And we say "tree" because the actual engines are wrapped in several layers:

- `Resolve` generates `conda.common.logic.Clauses` objects as needed.
- `Clauses` is a tight wrapper around its private `conda.common._logic.Clauses` counterpart. Let's call the former `_Clauses`. It simply wraps the `_Clauses` API with `._eval()` calls and other shortcuts for convenience.
- `_Clauses` provides an API to process the raw SAT formulas or clauses. It will wrap one of the `conda.common._logic._SatSolver` subclasses. *These* are the ones that wrap the SAT solver engines! So far, there are three subclasses, selectable via the `context.sat_solver` setting:
  - `_PycoSatSolver`, keyed as `pycosat`. This is the default one, a Python wrapper around the `picosat` project.
  - `_PySatSolver`, keyed as `pysat`. Uses the `Glucose4` solver found in the `pysat` project.
  - `_PyCryptoSatSolver`, keyed as `pycryptosat`. Uses the Python bindings for the CryptoMiniSat project.

In principle, more SAT solvers can be added to `conda` if a wrapper that subscribes to the `_SatSolver` API is used. However, if the reason is choosing a better performing engine, consider the following:

- The wrapped SAT solvers are already using compiled languages.
- Generating the clauses is indeed written in pure Python and has a non-trivial overhead.
- Optimization tricks like reducing the index and constraining the solution space have their costs if the "bets" were not successful.

**More about SAT solvers in general**

This guide did not cover the details of what SAT solvers are or do. If you want to read about them, consider checking the following resources:

- Aaron Meurer's slides about Conda internals. These slides reveal a lot of details of `conda` back in 2015. Some things have changed, but the core SAT solver behaviour is still well explained there.
- "Understanding and Improving Conda's performance"
- All the talks regarding solvers from Packaging-Con 2021. Check which talks belong to the Solvers track and enjoy!

## Logging

Logging in conda is based on the Python logging framework. In this guide, we'll describe the loggers and handlers used in conda, as well as how they are used.

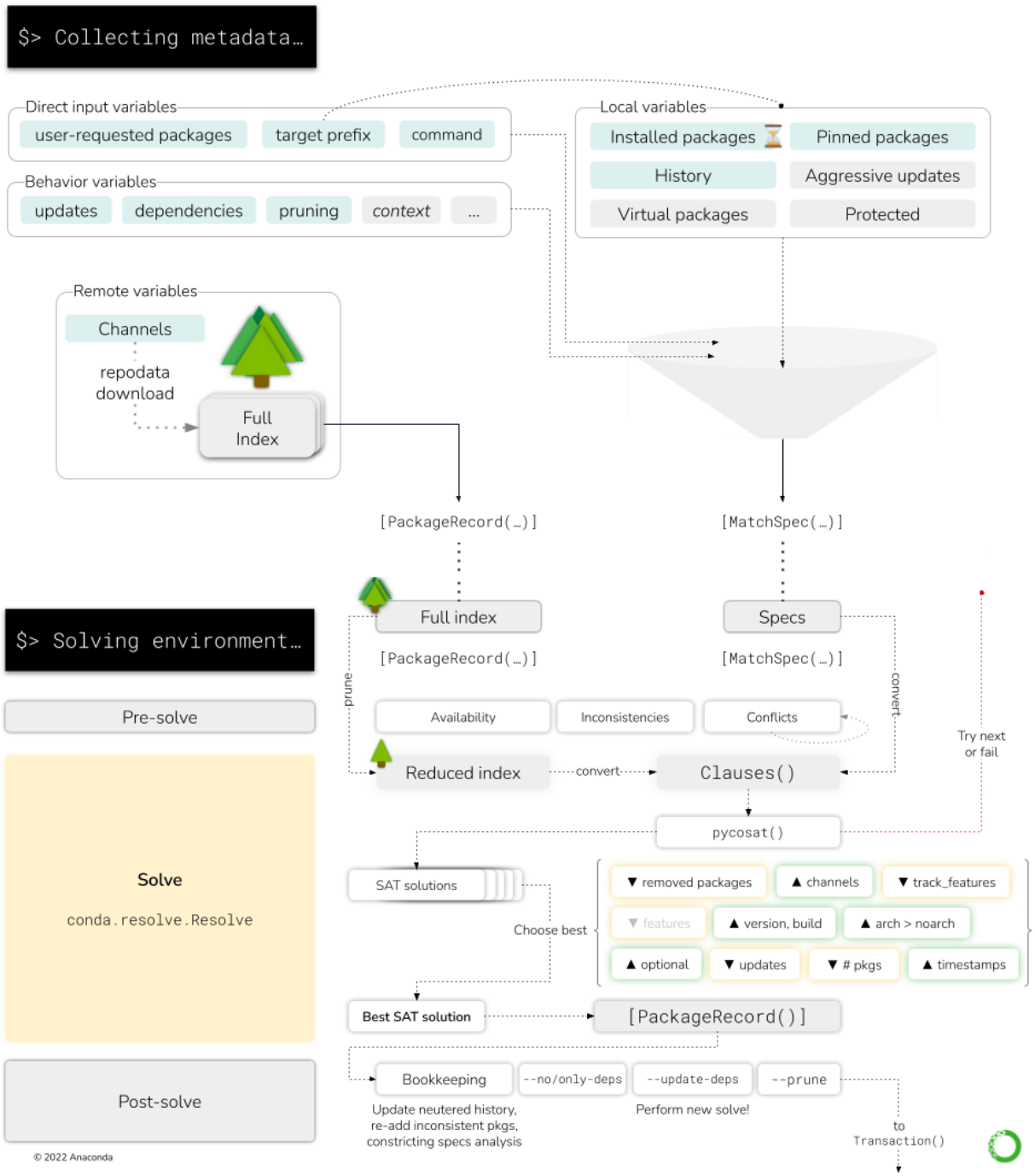



Fig. 4: Here you can see how the high level Solver API interacts with the low-level Resolve and Clauses objects.

The *Collecting metadata* step in the CLI report only compiles the necessary information from the CLI arguments, the prefix state and the chosen channels, presenting the SAT solver adapters with two critical pieces of information:

- The list of `MatchSpec` objects ("what the user wants in this environment")
- The list of `PackageRecord` objects ("the packages available in the channels")

So, in essence, the SAT solver takes the `MatchSpec` objects to select which `PackageRecord` objects satisfy the user request in the best way. The necessary computations are part of the "Solving environment…" step in the CLI report.

### Logger hierarchy

Python logging uses loggers as the source of logging messages. These loggers are organized as a single hierarchy defined by names given to individual loggers, with one nameless root logger. Other logger names are given as dot separated strings, such that, for example, the logger named `a.b.c` has the parent `a.b`, which in turn has the parent `a`, which in turn has the root logger as its parent. Python programs are free to use any name for loggers they like, but often the name will bear a relationship to the Python entity that is using it, most commonly there will be a module level logger, which is called `__name__`, i.e. `conda.<pkg>.<module>`, e.g. `conda.gateways.logging`. This approach naturally arranges loggers used in a single code base into a hierarchy that follows the package structure.

Conda largely follows this approach, however, it also makes use of some additional loggers.

Fig. 5: The conda logger hierarchy. Dotted lines represent relations with `propagate = False`.

The full hierarchy of all module level loggers is given below at full-module-loggers.

### The root logger

The root logger is not used directly as a logging target, but it is used as a building block in the configuration of the logging system.

### The `conda` subhierarchy

The `conda` subhierarchy consists of all loggers whose name starts with `conda.` and it is mostly configured via the `conda` logger itself in *`conda.gateways.logging.initialize_logging()`*.

Additionally, the following five loggers are used for other output. These are likely to be replaced and should not be used in new code.

- `conda.stdout`
- `conda.stderr`
- `conda.stdoutlog`
- `conda.stderrlog`
- `conda.stdout.verbose`

### Other loggers

Three more loggers appear in conda, namely `progress.update`and `progress.stop`, which only appear in `conda.plan.execute_actions`, which in turn is deprecated (c.f. https://github.com/conda/conda/pull/13881); and `auxlib` which likely is a remnant from before the auxlib code was completely absorbed into conda, since it only appears to be adorned with `conda.auxlib.NullHandler` in `conda.auxlib.__init__`.

#### Potential effect on other loggers

There are three other functions that use `logging.getLogger()` and hence might affect other loggers. They are *conda.gateways.logging.set_file_logging()* that is never used in the code base and the context managers *conda.common.io.disable_logger()* and *conda.common.io.stderr_log_level()*, which are only used in testing.

#### Root logger in auxlib

In *conda.auxlib.logz*, the root logger is modified.

### 4.6.5 Writing Tests

This section contains a series of guides and guidelines for writing tests in the `conda` repository.

---

#### Guides

*Integration Tests* This guide gives an overview of how to write integration tests using full command invocation. It also covers creating fixtures to use with these types of tests.

#### Integration Tests

Integration tests in `conda` test the application from a high level where each test can potentially cover large portions of the code. These tests may also use the local file system and/or perform network calls. In the following sections, we cover several examples of exactly how these tests look. When writing your own integration tests, these should serve as a good starting point.

##### `conda_cli` Fixture: Running CLI level tests

> **Note**
>
> The `conda_cli` fixture is a `scope="function"` fixture meaning it can only be used within tests or other `scope="function"` fixtures. For `"module"`, `"package"`, or `"session"` scoped fixtures use `session_conda_cli` instead.

CLI level tests are the highest level integration tests you can write. This means that the code in the test is executed as if you were running it from the command line. For example, you may want to write a test to confirm that an environment is created after successfully running `conda create`. A test like this would look like the following:

Listing 3: Integration test for `conda create`

```
from __future__ import annotations

import json
from typing import TYPE_CHECKING

if TYPE_CHECKING:
```

(continues on next page)

(continued from previous page)

```
    from pathlib import Path

    from conda.testing.fixtures import CondaCLIFixture

pytest_plugins = "conda.testing.fixtures"


def test_conda_create(conda_cli: CondaCLIFixture, tmp_path: Path):
    # setup, create environment
    out, err, code = conda_cli("create", "--prefix", tmp_path, "--yes")

    assert f"conda activate {tmp_path}" in out
    assert not err  # no errors
    assert not code  # success!

    # verify everything worked using the `conda env list` command
    out, err, code = conda_cli("env", "list", "--json")

    assert any(
        tmp_path.samefile(path)
        for path in json.loads(out).get("envs", [])
    )
    assert not err  # no errors
    assert not code  # success!

    # cleanup, remove environment
    out, err, code = conda_cli("remove", "--all", "--prefix", tmp_path)

    assert out
    assert not err  # no errors
    assert not code  # success!
```

Let's break down exactly what is going on in the code snippet above:

First, we rely on a fixture (`conda_cli`) that allows us to run a command using the current running process. This is much more efficient and quicker than running CLI tests via subprocesses.

In the test itself, we first create a new environment by effectively running `conda create`. This function returns the standard out, standard error, and the exit code of the command. This allows us to perform our inspections in order to determine whether the command ran successfully.

The second part of the test again uses the `conda_cli` fixture to call `conda env list`. This time, we pass the `--json` flag, which allows capturing JSON that we can better parse and more easily inspect. We then assert whether the environment we just created is actually in the list of environments available.

Finally, we destroy the environment we just created and ensure the standard error and the exit code are what we expect them to be.

> **Warning**
>
> It is preferred to use temporary directories (e.g., `tmp_path`) whenever possible for automatic cleanup after tests are run. Otherwise, remember to remove anything created during the test since it will be present when other tests are run and may result in unexpected race conditions.

### `tmp_env` Fixture: Creating a temporary environment

> **Note**
>
> The `tmp_env` fixture is a `scope="function"` fixture meaning it can only be used within tests or other `scope="function"` fixtures. For `"module"`, `"package"`, or `"session"` scoped fixtures use `session_tmp_env` instead.

The `tmp_env` fixture is a convenient way to create a temporary environment for use in tests:

Listing 4: Integration test for creating an environment with `numpy`

```python
from __future__ import annotations

from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from conda.testing.fixtures import CondaCLIFixture, TmpEnvFixture

pytest_plugins = "conda.testing.fixtures"


def test_environment_with_numpy(
    tmp_env: TmpEnvFixture,
    conda_cli: CondaCLIFixture,
):
    with tmp_env("numpy") as prefix:
        out, err, code = conda_cli("list", "--prefix", prefix)

        assert out
        assert not err  # no error
        assert not code  # success!
```

### `path_factory` Fixture: Creating a unique (non-existing) path

The `path_factory` fixture extends pytest's tmp_path fixture to provide unique, unused paths. This makes it easier to generate new paths in tests:

Listing 5: Integration test for renaming an environment

```python
from __future__ import annotations

from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from pathlib import Path

    from conda.testing.fixtures import (
        CondaCLIFixture,
        PathFactoryFixture,
        TmpEnvFixture,
```

(continues on next page)

(continued from previous page)

```
    )

pytest_plugins = "conda.testing.fixtures"


def test_conda_rename(
    path_factory: PathFactoryFixture,
    tmp_env: TmpEnvFixture,
    conda_cli: CondaCLIFixture,
    tmp_path: Path,
):
    # each call to `path_factory` returns a unique path
    assert path_factory() != path_factory()

    # each call to `path_factory` returns a path that is a child of `tmp_path`
    assert path_factory().parent == path_factory().parent == tmp_path

    with tmp_env() as prefix:
        out, err, code = conda_cli("rename", "--prefix", prefix, path_factory())

        assert out
        assert not err  # no error
        assert not code  # success!
```

### Tests with fixtures

Sometimes in integration tests, you may want to re-use the same type of environment more than once. Copying and pasting this setup and teardown code into each individual test can make these tests more difficult to read and harder to maintain.

To overcome this, `conda` tests make extensive use of `pytest` fixtures. Below is an example of the previously-shown test, except that we now make the focus of the test the `conda env list` command and move the creation and removal of the environment into a fixture:

Listing 6: Integration test for `conda create`

```
from __future__ import annotations

import json
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from pathlib import Path

    from conda.testing.fixtures import CondaCLIFixture, TmpEnvFixture

pytest_plugins = "conda.testing.fixtures"


@pytest.fixture
def env_one(tmp_env: TmpEnvFixture) -> Path:
```

(continues on next page)

(continued from previous page)

```
    with tmp_env() as prefix:
        yield prefix


def test_conda_create(env_one: Path, conda_cli: CondaCLIFixture):
    # verify everything worked using the `conda env list` command
    out, err, code = conda_cli("env", "list", "--json")

    assert any(
        env_one.samefile(path)
        for path in json.loads(out).get("envs", [])
    )
    assert not err  # no errors
    assert not code  # success!
```

In the fixture named `env_one`, we create a new environment using the `tmp_env` fixture. We yield to mark the end of the setup. Since the `tmp_env` fixture extends `tmp_path` no additional teardown is needed.

This fixture will be run using the default scope in `pytest`, which is `function`. This means that the setup and teardown will occur before and after each test that requests this fixture. If you need to share an environment or other pieces of data between tests, just remember to set the fixture scope appropriately. Read here for more information on `pytest` fixture scopes.

### Testing with Windows AppLocker

Windows environments with AppLocker enabled present unique challenges for `conda` development and testing. This guide explains how to set up a testing environment with AppLocker to ensure `conda` works correctly in these environments.

### Why Test with AppLocker?

AppLocker is Microsoft's application control solution that allows organizations to:

- Control which applications and files users can run
- Create rules to allow or deny applications from running based on file attributes
- Create exceptions to rules

Many enterprise environments use AppLocker to restrict script execution, which can impact environmnet activation and execution processes. Testing with AppLocker ensures `conda` works properly in these restricted environments.

### Setting Up AppLocker for Testing

#### Step 1: Enable the Application Identity Service

> **Note**
>
> The Application Identity Service is required for AppLocker to function properly.

1. Open the **Services** application (press `Win+R`, type `services.msc`, and press Enter)
2. Find **Application Identity** in the list of services
3. Right-click on it and select **Properties**
4. *Optional*: Change **Startup type** to **Automatic** if you want the service to start on boot
5. Click **Start** to start the service
6. Click **OK** to close the properties window

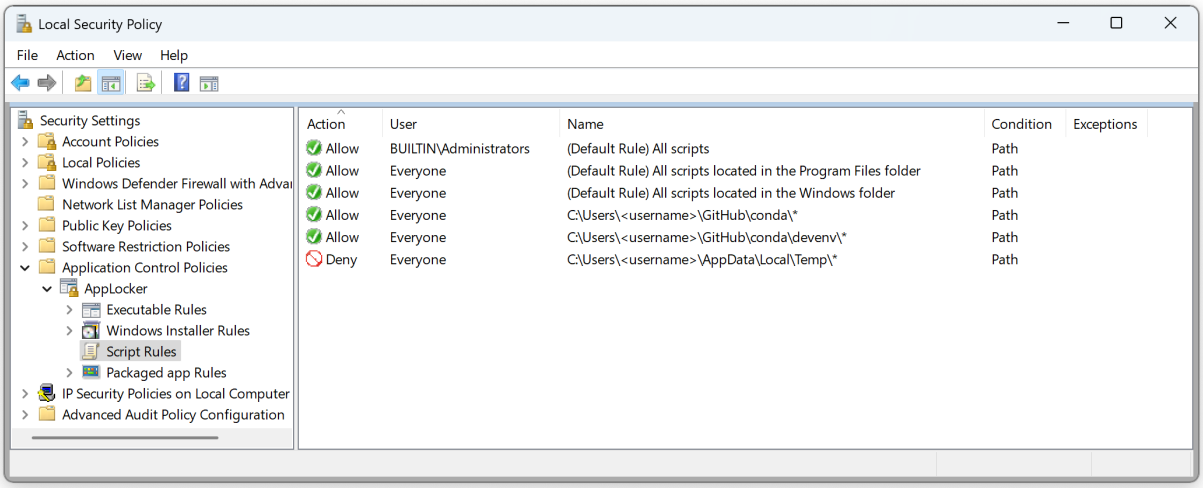
#### Step 2: Configure AppLocker Enforcement

1. Open **Local Security Policy** (press `Win+R`, type `secpol.msc`, and press Enter)
2. Navigate to **Security Settings > Application Control Policies > AppLocker**
3. Right-click on **AppLocker** and select **Properties**
4. Under the **Enforcement** tab, check **Script Rules** and set it to **Enforce rules**
5. Click **OK** to close the properties window

#### Step 3: Create AppLocker Rules

1. In the **Local Security Policy** window, navigate to **Script Rules** under **AppLocker**
2. Right-click on **Script Rules** and select **Create Default Rules** to establish baseline rules
3. Create an Allow Rule for your development environment:
    - Right-click on **Script Rules** and select **Create New Rule…**
    - Choose **Allow** under Permissions and set the user/group to **Everyone**
    - Select **Path** as the condition
    - Enter the path to your development environment (e.g., path to `devenv`)
    - Complete the wizard without adding exceptions
4. Create an Allow Rule for the conda source code location using the same process
5. Create a Deny Rule for the `%TEMP%` directory:
    - Follow the same process but choose **Deny** under Permissions
    - Set the absolute path
6. Restart your computer to apply the rules

---




## General Guidelines

- *Preferred test style (pytest)*
- *Organizing tests*
- *The "conda.testing" module*
- *Adding new fixtures*
- *The context object*

> **Note**
>
> It should be noted that existing tests may deviate from these guidelines, and that is okay. These guidelines are here to inform how we would like all new tests to look and function.

## Preferred test style (pytest)

Although our codebase includes class-based `unittest` tests, our preferred format for all new tests are `pytest` style tests. These tests are written using functions and handle the setup and teardown of context for tests using fixtures. We recommend familiarizing yourself with `pytest` first before attempting to write tests for `conda`. Head over to their Getting Started Guide to learn more.

## Organizing tests

Tests should be organized in a way that mirrors the main `conda` module. For example, if you were writing a test for a function in `conda/base/context.py`, you would place this test in `tests/base/test_context.py`.

### The "conda.testing" module

This is a module that contains anything that could possibly help with writing tests, including fixtures, functions, and classes. Feel free to make additions to this module as you see fit, but be mindful of organization. For example, if your testing utilities are primarily only for the `base` module considering storing these in `conda.testing.base`.

### Adding new fixtures

For fixtures that have a very limited scope or purpose, it is okay to define these alongside the tests themselves. However, if these fixtures could be used across multiple tests, then they should be saved in a separate `fixtures.py` file. The `conda.testing` module already contains several of these files.

If you want to add new fixtures within a new file, be sure to add a reference to this module in `tests/conftest.py::pytest_plugins`. This is our preferred way of making fixtures available to our tests. Because of the way these are included in the environment, you should be mindful of naming schemes and choose ones that likely will not collide with each other. Consider using a prefix to achieve this.

### The context object

The context object in `conda` is used as a singleton. This means that everytime the `conda` command runs, only a single object is instantiated. This makes sense as it holds all the configuration for the program and re-instantiating it or making multiple copies would be inefficient.

Where this causes problems is during tests where you may want to run `conda` commands potentially hundreds of times within the same process. Therefore, it is important to always reset this object to a fresh state when writing tests.

This can be accomplished by using the `reset_context` function, which also lives in the `conda.base.context` module. The following example shows how you would modify the context object and then reset it using the `reset_conda_context` `pytest` fixture:

```
import os
import tempfile

from conda.base.context import reset_context, context
from conda.testing.fixtures import reset_conda_context

TEST_CONDARC = """
channels:
  - test-channel
"""


def test_that_uses_context(reset_conda_context):
    # We first created a temporary file to hold our test configuration
    with tempfile.TemporaryDirectory() as tempdir:
        condarc_file = os.path.join(tempdir, "condarc")

        with open(condarc_file, "w") as tmp_file:
            tmp_file.write(TEST_CONDARC)

        # We use the reset_context function to load our new configuration
        reset_context(search_path=(condarc_file,))
```

(continues on next page)

(continued from previous page)

```
        # Run various test assertions, below is an example
        assert "test-channel" in context.channels
```

Using this testing fixture ensures that your context object is returned to the way it was before the test. For this specific test, it means that the `channels` setting will be returned to its default configuration. If you ever need to manually reset the context during a test, you can do so by manually invoking the `reset_context` command like in the following example:

```
from conda.base.context import reset_context


def test_updating_context_manually():
    # Load some custom variables into context here like above...

    reset_context()

    # Continue testing with a fresh context...
```

### 4.6.6 Deprecations

Conda abides by the Deprecation Schedule defined in CEP-9. To help make deprecations as much of a no-brainer as possible we provide several helper decorators and functions to facilitate the correct deprecation process.

**Tip**

The conda pending/deprecation versions abide by the following formula:

```
pending_dep = (the next release + 4 months) round up to nearest .3 or .9
deprecation = pending_dep + 6 months
```

Example:

```
pending_dep = (25.1 + 4 months) round up to nearest .3 or .9 = 25.9
deprecation = 25.9 + 6 months = 26.3
```

#### Functions, Methods, Properties, and Classes

**Warning**

To deprecate Enums treat them like constants (see *Constants and Enums*).

The simplest use case is for deprecating any function, method, or property:

Listing 7: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3")
```

(continues on next page)

(continued from previous page)

```
def bar():
    ...
```

Listing 8: Example invocation.

```
>>> import foo
>>> foo.bar()
<stdin>:1: PendingDeprecationWarning: foo.bar is pending deprecation and will be removed␣
↪in 24.3.
```

As a minimum we must always specify two versions:

1. the future deprecation release in which the function, method, or property will be marked as deprecated; prior to that the feature will show up as pending deprecation (which we treat as a commenting period), and
2. the subsequent deprecation release in which the function, method, or property will be removed from the code base.

Additionally, you may provide an addendum to inform the user what they should do instead:

Listing 9: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3", addendum="Use `qux` instead.")
def bar():
    ...
```

Listing 10: Example invocation.

```
>>> import foo
>>> foo.bar()
<stdin>:1: PendingDeprecationWarning: foo.bar is pending deprecation and will be removed␣
↪in 24.3. Use `qux` instead.
```

### Keyword Arguments

> **Warning**
>
> Deprecating or renaming a positional argument is unnecessarily complicated and is not supported. Instead, it is recommended to either (1) devise a custom way of detecting usage of a deprecated positional argument (e.g., type checking) and use the `conda.deprecations.deprecated.topic` function (see *Topics*) or (2) deprecate the function/method itself and define a new function/method without the deprecated argument.

Similarly to deprecating a function or method it is common to deprecate a keyword argument:

Listing 11: Example file, `foo.py`.

```
from conda.deprecations import deprecated
```

(continues on next page)

(continued from previous page)

```
# prior implementation
# def bar(is_true=True):
#     ...


@deprecated.argument("23.9", "24.3", "is_true")
def bar():
    ...
```

Listing 12: Example invocation.

```
>>> import foo
>>> foo.bar(is_true=True)
<stdin>:1: PendingDeprecationWarning: foo.bar(is_true) is pending deprecation and will␣
↪be removed in 24.3.
```

Or to rename the keyword argument:

Listing 13: Example file, `foo.py`.

```
from conda.deprecations import deprecated


# prior implementation
# def bar(is_true=True):
#     ...


@deprecated.argument("23.9", "24.3", "is_true", rename="enabled")
def bar(enabled=True):
    ...
```

Listing 14: Example invocation.

```
>>> import foo
>>> foo.bar(is_true=True)
<stdin>:1: PendingDeprecationWarning: foo.bar(is_true) is pending deprecation and will␣
↪be removed in 24.3. Use 'enabled' instead.
```

#### argparse.Action

Occasionally, there is a need to deprecate CLI arguments. For this, we provide a helper function to monkeypatch any `argparse.Action`:

Listing 15: Example file, `foo.py`.

```
import argparse
from conda.deprecations import deprecated

parser = argparse.ArgumentParser()
```

(continues on next page)

(continued from previous page)

```
parser.add_argument(
    "--force",
    dest="yes",
    action=deprecated.action(
        "23.9",
        "24.3",
        argparse._StoreTrueAction,
        addendum="Use `--yes` instead.",
    ),
    default=False,
)
parser.parse_args()
```

```
python foo.py --force
foo.py:16: PendingDeprecationWarning: `--force` is pending deprecation and will be␣
↪removed in 24.3. Use `--yes` instead.
```

### Constants and Enums

We also offer a way to deprecate global variables or constants:

Listing 16: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.constant("23.9", "24.3", "ULTIMATE_CONSTANT", 42)
```

Listing 17: Example invocation.

```
>>> import foo
>>> foo.ULTIMATE_CONSTANT
<stdin>:1: PendingDeprecationWarning: foo.ULTIMATE_CONSTANT is pending deprecation and␣
↪will be removed in 24.3.
```

Enums work similarly:

Listing 18: Example file, `foo.py`.

```
from enum import Enum
from conda.deprecations import deprecated

class Bar(Enum):
    ULTIMATE_CONSTANT = 42

deprecated.constant("23.9", "24.3", "Bar", Bar)
del Bar
```

Listing 19: Example invocation.

```
>>> from foo import Bar
<stdin>:1: PendingDeprecationWarning: foo.Bar is pending deprecation and will be removed␣
```

(continues on next page)

(continued from previous page)

```
↪in 24.3.
```

**Note**

Constants deprecation relies on the module's `__getattr__` introduced in PEP-562.

### Modules

Entire modules can be also be deprecated:

Listing 20: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.module("23.9", "24.3")
```

Listing 21: Example invocation.

```
>>> import foo
<stdin>:1: PendingDeprecationWarning: foo is pending deprecation and will be removed in␣
↪24.3.
```

### Topics

Finally, there are a multitude of other ways in which code may be run that also needs to be deprecated. To this end we offer a general purpose deprecation function:

Listing 22: Example file, `foo.py`.

```
from conda.deprecations import deprecated

def bar(...):
    # some logic

    if ...:
        deprecated.topic("23.9", "24.3", topic="The <TOPIC>")

    # some more logic
```

Listing 23: Example invocation.

```
>>> import foo
>>> foo.bar(...)
<stdin>:1: PendingDeprecationWarning: The <TOPIC> is pending deprecation and will be␣
↪removed in 24.3.
```

### 4.6.7 Releasing

Conda's releases may be performed via the rever command. Rever is configured to perform the activities for a typical conda release. To cut a release, simply run `rever <X.Y.Z>` where `<X.Y.Z>` is the release number that you want bump to. For example, `rever 1.2.3`.

However, it is always good idea to make sure that the you have permissions everywhere to actually perform the release. So it is customary to run `rever check` before the release, just to make sure.

The standard workflow is thus:

```
$ rever check
$ rever 1.2.3
```

If for some reason a release fails partway through, or you want to claw back a release that you have made, rever allows you to undo activities. If you find yourself in this pickle, you can pass the `--undo` option a comma-separated list of activities you'd like to undo. For example:

```
$ rever --undo tag,changelog,authors 1.2.3
```

Happy releasing!

### 4.6.8 Plugins

As of version `22.11.0`, conda has support for user plugins, enabling extension and/or alterations to some of its functionality.

#### Quick start

This is an example of a minimal working conda plugin that defines a new subcommand:

Listing 24: example_plugin.py

```
import conda.plugins
from conda.base.context import context


def command(arguments: list[str]):
    print("Conda subcommand!")


@conda.plugins.hookimpl
def conda_subcommands():
    yield conda.plugins.CondaSubcommand(
```

(continues on next page)

(continued from previous page)

```
        name="example",
        action=command,
        summary="Example of a conda subcommand",
    )
```

Let's break down what's going on here step-by-step:

1. First, we create the function `command` that serves as our subcommand. This function is passed a list of arguments which equal to `sys.argv[2:]`.
2. Next, we register this subcommand by using the `conda_subcommands` plugin hook. We do this by creating a function called `conda_subcommands` and then decorating it with `conda.plugins.hookimpl`.
3. The object we return from this function is `conda.plugins.CondaSubcommand`, which does several things:
    1. **name** is what we use to call this subcommand via the command line (i.e. "conda example")
    2. **action** is the function that will be called when we invoke "conda example"
    3. **summary** is the description of the of the subcommand that appears when users call "conda --help"

In order to actually use conda plugins, they must be packaged as Python packages. Furthermore, we also need to take advantage of a feature known as Python package entrypoints. We can define our Python package and the entry points by either using a `pyproject.toml` file (preferred) or a `setup.py` (legacy) for our project:

Listing 25: pyproject.toml

```
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"

[project]
name = "conda-example-plugin"
version = "1.0.0"
description = "Example conda plugin"
requires-python = ">=3.9"
dependencies = ["conda"]

[project.entry-points."conda"]
conda-example-plugin = "example_plugin"
```

Listing 26: setup.py

```
from setuptools import setup

setup(
    name="conda-example-plugin",
    install_requires="conda",
    entry_points={"conda": ["conda-example-plugin = example_plugin"]},
    py_modules=["example_plugin"],
)
```

In both examples shown above, we define an entry point for conda. It's important to make sure that the entry point is for "conda" and that it points to the correct module in your plugin package. Our package only consists a single Python module called `example_plugin`. If you have a large project, be sure to always point the entry point to the module containing the plugin hook declarations (i.e. where `conda.plugins.hookimpl` is used). We recommend using the

*plugin* submodule in these cases, e.g. *large_project.plugin* (in *large_project/plugin.py*).

### More examples

To see more examples of conda plugins, please visit the following resources:

- conda-plugins-template: This is a repository with full examples that could be used a starting point for your plugin

### Using other plugin hooks

For examples of how to use other plugin hooks, please read their respective documentation pages:

### Auth Handlers

The auth handlers plugin hook allows plugin authors to enable new modes of authentication within conda. Registered auth handlers will be available to configure on a per channel basis via the `channel_settings` configuration option in the `.condarc` file.

Auth handlers are subclasses of the *ChannelAuthBase* class, which is itself a subclass of requests.auth.AuthBase. The *ChannelAuthBase* class adds an additional `channel_name` property to the requests.auth.AuthBase class. This is necessary for appropriate handling of channel based authentication in conda.

For more information on how to implement your own auth handlers, please read the requests documentation on Custom Authentication.

**class CondaAuthHandler**

Return type to use when the defining the conda auth handlers hook.

**Parameters**

- **name** -- Name (e.g., `basic-auth`). This name should be unique and only one may be registered at a time.
- **handler** -- Type that will be used as the authentication handler during network requests.

**handler**

**name**

**conda_auth_handlers()**

Register a conda auth handler derived from the requests API.

This plugin hook allows attaching requests auth handler subclasses, e.g. when authenticating requests against individual channels hosted at HTTP/HTTPS services.

**Example:**

```
import os
from conda import plugins
from requests.auth import AuthBase


class EnvironmentHeaderAuth(AuthBase):
    def __init__(self, *args, **kwargs):
        self.username = os.environ["EXAMPLE_CONDA_AUTH_USERNAME"]
        self.password = os.environ["EXAMPLE_CONDA_AUTH_PASSWORD"]
```

(continues on next page)

(continued from previous page)

```
    def __call__(self, request):
        request.headers["X-Username"] = self.username
        request.headers["X-Password"] = self.password
        return request


@plugins.hookimpl
def conda_auth_handlers():
    yield plugins.CondaAuthHandler(
        name="environment-header-auth",
        handler=EnvironmentHeaderAuth,
    )
```

### Environment Specifiers

Conda can create environments from several file formats. Currently, conda natively supports creating environments from:

- A *YAML environment.yaml file*
- An *"explicit" text file*

For more information on how to manage conda environments, see the **`Managing environments`_** documentation.

### Example plugin

The available readers can be extended with additional plugins via the `conda_environment_specifiers` hook.

> **Hint**
>
> To see a fully functioning example of a Environment Spec backend, checkout the *`yaml_file`* module.

**`conda_environment_specifiers`**()

**EXPERIMENTAL** Register new conda env spec type

The example below defines a type of conda env file called "random". It can parse a file with the file extension *.random*. This plugin will ignore whatever is in the input environment file and produce an environment with a random name and with random packages.

**Example:**

```
import json
import random
from pathlib import Path
from subprocess import run
from conda import plugins
from ...plugins.types import EnvironmentSpecBase
from conda.env.env import Environment

packages = ["python", "numpy", "scipy", "matplotlib", "pandas", "scikit-learn"]
```

(continues on next page)

(continued from previous page)

```
class RandomSpec(EnvironmentSpecBase):
    extensions = {".random"}

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        # Return early if no filename was provided
        if self.filename is None:
            return False

        # Extract the file extension (e.g., '.txt' or '' if no extension)
        file_ext = os.path.splitext(self.filename)[1]

        # Check if the file has a supported extension and exists
        return any(
            spec_ext == file_ext and os.path.exists(self.filename)
            for spec_ext in RandomSpec.extensions
        )

    def env(self):
        return Environment(
            name="".join(random.choice("0123456789abcdef") for i in range(6)),
            dependencies=[random.choice(packages) for i in range(6)],
        )


@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
    )
```

### Defining `EnvironmentSpecBase`

The first class we define is a subclass of *`EnvironmentSpecBase`*. The base class is an abstract base class which requires us to define our own implementations of its abstract methods:

- `can_handle` Determines if the defined plugin can read and operate on the provided file.
- `env` Expresses the provided environment file as a conda environment object.

The class may also define the boolean class variable *detection_supported*. When set to `True`, the plugin will be included in the environment spec type discovery process. Otherwise, the plugin will only be able to be used when it is specifically selected. By default, this value is `True`.`

Be sure to be very specific when implementing the `can_handle` method. It should only return a `True` if the file can be parsed by the plugin. Making the `can_handle` method too permissive in the types of files it handles may lead to conflicts with other plugins. If multiple installed plugins are able to `can_handle` the same file type, conda will return an error to the user.

### Registering the plugin hook

In order to make the plugin available to conda, it must be registered with the plugin manager. Define a function with the `plugins.hookimpl` decorator to register our plugin which returns our class wrapped in a *CondaEnvironmentSpecifier* object. Note, that by default autodetection is enabled.

```
@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
    )
```

### Using the Plugin

Once this plugin is registered, users will be able to create environments from the types of files specified by the plugin. For example to create a *random* environment using the plugin defined above:

```
conda env create --file /doesnt/matter/any/way.random
```

### Plugin detection

When conda is trying to determine which environment spec plugin to use it will loop through all registered plugins and call their `can_handle` function. If one (and only one) plugin returns a `True` value, conda will use that plugin to read the provided environment spec. However, if multiple plugins are detected an error will be raised.

Plugin authors may explicitly disable their plugin from being detected by disabling autodetection in their plugin class

```
class RandomSpec(EnvironmentSpecBase):
    detection_supported = False

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        return True

    def env(self):
        return Environment(name="random-environment", dependencies=["python", "numpy"])
```

End users can bypass environment spec plugin detection and explicitly request a plugin to be used by configuring conda to use a particular installed plugin. This can be done by either:

- cli by providing the `--env-spec` flag, or
- environment variable by setting the `CONDA_ENV_SPEC` environment variable, or
- `.condarc` by setting the `environment_specifier` config field

### Another example plugin

In this example, we want to build a more realistic environemnt spec plugin. This plugin has a scheme which expresses what it expects a valid environment file to contain. In this example, a valid environment file is a `.json` file that defines:

- an environment name (required)
- a list of conda dependencies

```
import os
from pydantic import BaseModel

from conda.plugins import CondaEnvironmentSpecifier, hookimpl
from conda.plugins.types import EnvironmentSpecBase
from conda.env.env import Environment


class MySimpleEnvironment(BaseModel):
    """An model representing an environment file."""

    # required
    name: str

    # optional
    conda_deps: list[str] = []


class MySimpleSpec(EnvironmentSpecBase):
    def __init__(self, filename=None):
        self.filename = filename

    def _parse_data(self) -> MySimpleEnvironment:
        """ "Validate and convert the provided file into a MySimpleEnvironment"""
        with open(self.filename, "rb") as fp:
            json_data = fp.read()

        return MySimpleEnvironment.model_validate_json(json_data)

    def can_handle(self) -> bool:
        """
        Validates loader can process environment definition.
        This can handle if:
              * the file exists
              * the file can be read
              * the data can be parsed as JSON into a MySimpleEnvironment object

        :return: True if the file can be parsed and handled, False otherwise
        """
        if not os.path.exists(self.filename):
            return False
        try:
            self._parse_data()
        except Exception:
            return False
```

(continues on next page)

(continued from previous page)

```
        return True

    @property
    def env(self) -> Environment:
        """Returns the Environment representation of the environment spec file"""
        data = self._parse_data()
        return Environment(
            name=data.name,
            dependencies=data.conda_deps,
        )


@hookimpl
def conda_environment_specifiers():
    yield CondaEnvironmentSpecifier(
        name="mysimple",
        environment_spec=MySimpleSpec,
    )
```

We can test this out by trying to create a conda environment with a new file that is compatible with the definied spec. Create a file `testenv.json`

```
{
   "name": "mysimpletest",
   "conda_deps": ["numpy", "pandas"]
}
```

Then, create the environment

```
$ conda env create --file testenv.json
```

### Health Checks

Conda doctor can be extended with the `health_checks` plugin hook. Write new health checks using the `health_checks` plugin hook, install the plugins you wrote and they will run every time `conda doctor` command is run. The `action` function is where you specify the code you want to be executed with `conda doctor`.

**class CondaHealthCheck**

Return type to use when defining conda health checks plugin hook.

**action**

**name**

**conda_health_checks()**

Register health checks for conda doctor.

This plugin hook allows you to add more "health checks" to conda doctor that you can write to diagnose problems in your conda environment. Check out the health checks already shipped with conda for inspiration.

**Example:**

```
from conda import plugins


def example_health_check(prefix: str, verbose: bool):
    print("This is an example health check!")


@plugins.hookimpl
def conda_health_checks():
    yield plugins.CondaHealthCheck(
        name="example-health-check",
        action=example_health_check,
    )
```

### Post-commands

Conda commands can be extended with the `conda_post_commands` plugin hook. By specifying the set of commands you would like to use in the `run_for` configuration option, you can execute code via the `action` option after these commands are run. The functions are provided a `command` argument representing the name of the command currently running. If the command fails for any reason, this plugin hook will not be run.

If you would like to target `conda env` commands, prefix the command name with `env_`. For example, `conda env list` would be passed to `run_for` as `env_list`.

**class CondaPostCommand**

Return type to use when defining a conda post-command plugin hook.

For details on how this is used, see *conda_post_commands()*.

**Parameters**

- **name** -- Post-command name (e.g., `custom_plugin_post_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

**action**

**name**

**run_for**

**conda_post_commands()**

Register post-command functions in conda.

**Example:**

```
from conda import plugins


def example_post_command(command):
    print("post-command action")


@plugins.hookimpl
```

(continues on next page)

(continued from previous page)

```
def conda_post_commands():
    yield plugins.CondaPostCommand(
        name="example-post-command",
        action=example_post_command,
        run_for={"install", "create"},
    )
```

### Pre-commands

Conda commands can be extended with the `conda_pre_commands` plugin hook. By specifying the set of commands you would like to use in the `run_for` configuration option, you can execute code via the `action` option before these commands are run. The functions are provided a `command` argument representing the name of the command currently running.

If you would like to target `conda env` commands, prefix the command name with `env_`. For example, `conda env list` would be passed to `run_for` as `env_list`.

**class CondaPreCommand**

Return type to use when defining a conda pre-command plugin hook.

For details on how this is used, see *conda_pre_commands()*.

**Parameters**

- **name** -- Pre-command name (e.g., `custom_plugin_pre_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

**action**

**name**

**run_for**

**conda_pre_commands()**

Register pre-command functions in conda.

**Example:**

```
from conda import plugins


def example_pre_command(command):
    print("pre-command action")


@plugins.hookimpl
def conda_pre_commands():
    yield plugins.CondaPreCommand(
        name="example-pre-command",
        action=example_pre_command,
        run_for={"install", "create"},
    )
```

### Pre-transactions

*Solver transactions* can be extended with the `conda_pre_transaction_actions` plugin hook. This plugin hook accepts a subclass of `Action` which will be instantiated and prepended to conda's transaction action list. When defining the class you must define the following methods:

- `execute`: this is the primary place to put code you wish to run during the action.
- `verify`: this is run before the action is executed. This is a good place to check for conditions that would cause the action to fail.
- `cleanup`: this is run after the action is executed. This is a good place to clean up any resources that were created during the action.
- `reverse`: in the case of a failure, this allows you to define any reversal procedures.

**class CondaPreTransactionAction**

Return type to use when defining a pre-transaction action hook.

For details on how this is used, see *conda_pre_transaction_actions()*.

**Parameters**

- **name** -- Pre transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**action**

**name**

**conda_pre_transaction_actions()**

Register pre-transaction hooks.

Pre-transaction hooks run before all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

**Example:**

```
from conda import plugins
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )
```

(continues on next page)

(continued from previous page)

```
    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_pre_transaction_actions(
        self,
    ) -> Iterable[plugins.CondaPreTransactionAction]:
        yield plugins.CondaPreTransactionAction(
            name="example-pre-transaction-action",
            action=PrintAction,
        )
```

### Post-transactions

*Solver transactions* can be extended with the `conda_post_transaction_actions` plugin hook. This plugin hook accepts a subclass of `Action` which will be instantiated and appended to the end of conda's transaction action list. When defining the class you must define the following methods:

- `execute`: this is the primary place to put code you wish to run during the action.
- `verify`: this is run before the action is executed. This is a good place to check for conditions that would cause the action to fail.
- `cleanup`: this is run after the action is executed. This is a good place to clean up any resources that were created during the action.
- `reverse`: in the case of a failure, this allows you to define any reversal procedures.

**class CondaPostTransactionAction**

Return type to use when defining a post-transaction action hook.

For details on how this is used, see *conda_post_transaction_actions()*.

**Parameters**

- **name** -- Post transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**action**

**name**

**conda_post_transaction_actions()**

Register post-transaction hooks.

Post-transaction hooks run after all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

**Example:**

```python
from conda import plugins
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_post_transaction_actions(
        self,
    ) -> Iterable[plugins.CondaPostTransactionAction]:
        yield plugins.CondaPostTransactionAction(
            name="example-post-transaction-action",
            action=PrintAction,
        )
```

### `PrefixData` loaders

The `PrefixData` class exposes the contents of a given conda environment (prefix) as a series of `PrefixRecord` objects. This plugin hook allows users to write logic to expose non-conda packages as conda-like metadata. This can be useful to allow `conda list` to report additional ecosystems, like PyPI or others.

**`class CondaPrefixDataLoader`**

Define new loaders to expose non-conda packages in a given prefix as `PrefixRecord` objects.

**Parameters**

- **`name`** -- name of the loader
- **`loader`** -- a function that takes a prefix and a dictionary that maps package names to `PrefixRecord` objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

**loader**

**name**

**conda_prefix_data_loaders**()

Register new loaders for PrefixData

The example below defines how to expose the packages installed by a hypothetical 'penguin' tool as conda packages.

**Example:**

```
from pathlib import Path

from conda import plugins
from conda.common.path import PathType
from conda.models.records import PrefixRecord
from conda.plugins.types import CondaPrefixDataLoader


@plugins.hookimpl
def conda_prefix_data_loaders():
    yield CondaPrefixDataLoader(
        "hypothetical",
        load_hypothetical_packages,
    )


def load_hypothetical_packages(
    prefix: PathType, records: dict[str, PrefixRecord]
) -> dict[str, PrefixRecord]:
    penguin_records = {}
    for info in Path(prefix).glob("lib/penguin/*.penguin-info"):
        name, version = info.name.rsplit("-", 1)
        kwargs = {}  # retrieve extra fields here
        penguin_records[name] = PrefixRecord(
            name=name, version=version, build_number=0, build="0", **kwargs
        )
    records.update(penguin_records)
    return penguin_records
```

### Reporter Backends

Reporter backends is a plugin hook that allows you to customize the display of certain elements at the command line. Below is a list of all the elements that can currently be customized:

- Progress bar (displayed during package downloads)
- Spinner (displayed while conda is waiting for a long running task to finish)
- Confirmation input (yes/no dialogs)
- `conda info` layout

This can be configured using the `console` setting either as a command line option or by defining it in the `.condarc` file.

For more information on configuring and using reporter backends in conda itself see:

- *Settings: console*

### Example plugin

The following example defines a simple plugin which uses the `pformat()` function for rendering parts of conda's output:

> **Hint**
>
> This is just a partial example. To see a fully functioning example of a reporter backend, checkout the *console* module.

```
from pprint import pformat

from conda import plugins
from conda.plugins.types import (
    CondaReporterBackend,
    ReporterRendererBase,
    ProgressBarBase,
)


class PprintReporterRenderer(ReporterRendererBase):
    """
    Implementation of the ReporterRendererBase abstract base class
    """

    def detail_view(self, data):
        return pformat(data)

    def envs_list(self, data):
        formatted_data = pformat(data)
        return f"Environments: {formatted_data}"

    def progress_bar(self, description, io_context_manager) -> ProgressBarBase:
        """Returns our custom progress bar implementation"""
        return PprintProgressBar(description, io_context_manager)


class PprintProgressBar(ProgressBarBase):
    """
    Blank implementation of ProgressBarBase which does nothing.
    """

    def update_to(self, fraction) -> None:
        pass

    def refresh(self) -> None:
        pass
```

(continues on next page)

(continued from previous page)

```
    def close(self) -> None:
        pass


@plugins.hookimpl
def conda_reporter_backends():
    yield CondaReporterBackend(
        name="pprint",
        description="Reporter backend based on the pprint module",
        renderer=PprintReporterRenderer,
    )
```

Below is a summary of everything we've defined:

### Defining `ReporterRendererBase`

The first class we define, `PprintReporterRenderer`, is a subclass of *`ReporterRendererBase`*. The base class is an abstract base class which requires us to define our own implementations of its abstract methods. These abstract methods are used by conda when rendering output and are where all the customization we want to do occurs.

### Defining `ProgressBarBase`

The second class we define is `PprintProgressBar`. For this example, it is just an empty implementation of the *`ProgressBarBase`*. Defining this effectively hides the progress bar when this reporter backend is configured. We do this in this tutorial because a full implementation would take too long to explain. Please check out *`TQDMProgressBar`* for a more realistic example using the tqdm library.

### Registering the plugin hook

Finally, we define the `conda_reporter_backends` function with the `plugins.hookimpl` decorator to register our plugin which returns the `PprintReporterRenderer` class wrapped in a *`CondaReporterBackend`* object. By registering it with `name` set to `pprint`, we will be able to reference this plugin as a new backend for the `console` setting.

### Using the reporter backend

To use our newly registered reporter backend, it can either be specified in our `.condarc` configuration file:

```
console: pprint
```

Or, it can be specified at the command line using the `--console` option:

```
conda info --envs --console=pprint
```

### Further reading

For detailed information on how to create a conda plugin from scratch, please see the following repository which also contains a cookiecutter recipe you can use to easily bootstrap your project:

- conda-plugins-template

Below are relevant areas of the API docs for the reporter backends plugin hook:

- *CondaReporterBackend* metadata object that must be returned from the reporter backends hook definition.
- *conda_reporter_backends()* hookspec definition for reporter backends which contains an example of its usage.
- *console* our default implementation for the `console` reporter backend.
- *json* our default implementation for the `json` reporter backend.

### Request Headers

The request headers plugin hooks allows plugin authors to add custom HTTP headers to HTTP requests within conda. This works by inserting these headers while preparing the requests.PreparedRequest object when calling requests.Session.request.

There are two hooks available for this purpose:

1. *conda_session_headers()*: This hook is used to define the custom headers that will be submitted with all HTTP requests (or a subset of hosts)
2. *conda_request_headers()*: This hook is used to define the custom headers that will be submitted with a specific HTTP request.

> **Warning**
>
> While the second hook (*conda_request_headers()*) can also create session headers, it is recommended to use the first hook (*conda_session_headers()*) for improved caching performance.

**conda_request_headers**(*host: str*, *path: str*)

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com` and a `path/to/endpoint.json` path.

**Example:**

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}
ENDPOINT = "/path/to/endpoint.json"


@plugins.hookimpl
def conda_request_headers(host: str, path: str):
    if host in HOSTS and path == ENDPOINT:
        yield plugins.CondaRequestHeader(
            name="Example-Header",
```

(continues on next page)

(continued from previous page)

```
            value="example",
        )
```

**conda_session_headers**(*host: str*)

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com`.

**Example:**

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}


@plugins.hookimpl
def conda_session_headers(host: str):
    if host in HOSTS:
        yield plugins.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

Both hook use the *CondaRequestHeader* class to define headers:

**class CondaRequestHeader**

Define vendor specific headers to include HTTP requests

For details on how this is used, see *conda_request_headers()* and *conda_session_headers()*.

**Parameters**

- **name** -- name of the header used in the HTTP request
- **value** -- value of the header used in the HTTP request

**name**

**value**

## Settings

The settings plugin hook allows plugin authors to add new settings to conda. Users will be able to use these new parameters either in `.condarc` files or define them as environment variables. For more information on configuration in conda, see *Configuration*.

The plugin hooks relies on using the various *conda.common.configuration.Parameter* sub-classes (e.g. *conda.common.configuration.PrimitiveParameter* or *conda.common.configuration.SequenceParameter*). For more examples of how these parameter classes are used, please see the *conda.base.context.Context* class.

**class CondaSetting**

Return type to use when defining a conda setting plugin hook.

For details on how this is used, see *conda_settings()*.

**Parameters**

- **name** -- name of the setting (e.g., `config_param`)

- **description** -- description of the setting that should be targeted towards users of the plugin
- **parameter** -- Parameter instance containing the setting definition
- **aliases** -- alternative names of the setting

**aliases**

**description**

**name**

**parameter**

**conda_settings**()

Register new setting

The example below defines a simple string type parameter

**Example:**

```
from conda import plugins
from conda.common.configuration import PrimitiveParameter, SequenceParameter


@plugins.hookimpl
def conda_settings():
    yield plugins.CondaSetting(
        name="example_option",
        description="This is an example option",
        parameter=PrimitiveParameter("default_value", element_type=str),
        aliases=("example_option_alias",),
    )
```

### Solvers

The conda solvers can be extended with additional backends with the `conda_solvers` plugin hook. Registered solvers will be available for configuration with the `solver` configuration and `--solver` command line option.

**class CondaSolver**

Return type to use when defining a conda solver plugin hook.

For details on how this is used, see *conda_solvers()*.

**Parameters**

- **name** -- Solver name (e.g., `custom-solver`).
- **backend** -- Type that will be instantiated as the solver backend.

**backend**

**name**

**conda_solvers**()

Register solvers in conda.

**Example:**

```python
import logging

from conda import plugins
from conda.core import solve

log = logging.getLogger(__name__)


class VerboseSolver(solve.Solver):
    def solve_final_state(self, *args, **kwargs):
        log.info("My verbose solver!")
        return super().solve_final_state(*args, **kwargs)


@plugins.hookimpl
def conda_solvers():
    yield plugins.CondaSolver(
        name="verbose-classic",
        backend=VerboseSolver,
    )
```

**Returns**
An iterable of solver entries.

### Subcommands

The conda CLI can be extended with the `conda_subcommands` plugin hook. Registered subcommands will be available under the `conda <subcommand>` command.

**class CondaSubcommand**

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

**Parameters**

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

**action**

**configure_parser**

**name**

**summary**

**conda_subcommands()**

Register external subcommands in conda.

**Example:**

```python
from conda import plugins


def example_command(args):
    print("This is an example command!")


@plugins.hookimpl
def conda_subcommands():
    yield plugins.CondaSubcommand(
        name="example",
        summary="example command",
        action=example_command,
    )
```

**Returns**
An iterable of subcommand entries.

### Virtual Packages

Conda allows for the registering of virtual packages in the index data via the plugin system. This mechanism lets users write plugins that provide version identification for properties only known at runtime (e.g., OS information).

**class CondaVirtualPackage**

Return type to use when defining a conda virtual package plugin hook.

For details on how this is used, see *conda_virtual_packages()*.

**Parameters**

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).
- **build** -- Virtual package build string (e.g., `x86_64`).

**build**

**name**

**to_virtual_package()**

**version**

**conda_virtual_packages()**

Register virtual packages in Conda.

**Example:**

```python
from conda import plugins


@plugins.hookimpl
def conda_virtual_packages():
    yield plugins.CondaVirtualPackage(
        name="my_custom_os",
```

(continues on next page)

(continued from previous page)

```
        version="1.2.3",
        build="x86_64",
    )
```

**Returns**
An iterable of virtual package entries.

### More information about how plugins work

Plugins in `conda` are implemented with the use of Pluggy, a Python framework used by other projects, such as `pytest`, `tox`, and `devpi`. `pluggy` provides the ability to extend and modify the behavior of `conda` via function hooking, which results in plugin systems that are discoverable with the use of Python package entrypoints.

For more information about how it works, we suggest heading over to their documentation.

### API

For even more detailed information about our plugin system, please the see the *Plugin API* section.

### Error handling

Errors in `conda` are routed through *`conda.exception_handler.ExceptionHandler`*, which can print additional information about the `conda` installation when an *unexpected* exception is found. These automatic reports can be really verbose and can get in the way of communicating *expected* errors. See this issue in conda-build as an example.

To mark exceptions as *expected*, plugins should raise *`conda.CondaError`* or a subclass thereof. See conda_auth.exceptions for an example.

### A note on licensing

For more information on which license to use for your custom plugin, please reference the "Choose an Open Source License" site. If you need help figuring out exactly which one to use, we advise communicating with a qualified legal professional.

## 4.6.9 Specifications

This section contains an incomplete list of conda specifications that may or may not be related to Conda Enhancement Proposals.

**Work in progress**

This page of the documentation is not yet finished and only contains a draft of the content.

### Technical specification: solver state

> **Note**
>
> This document is a technical specification, which might not be the best way to learn about how the solver works. For that, refer to *conda install* and *Solvers*.

The `Solver` API will pass a collection of `MatchSpec` objects (from now on, we will refer to them as `specs`) to the underlying SAT solver. How this list is built from the prefix state and context options is not a straightforward process, but an elaborate logic. This is better understood if we examine the *ingredients* that participate in the construction of `specs`. We will label them like this:

These groups below will *not* change during the solver attempts:

1. `requested`: `MatchSpec` objects the user is *explicitly* asking for.
2. `installed`: Installed packages, expressed as `PrefixRecord` objects. Empty if the environment did not exist.
3. `history`: Specs asked in the past: the `History`. Empty if the environment did not exist.
4. `aggressive_updates`: Packages included in the *aggressive updates* list. These packages are always included in any requests to make sure they stay up-to-date under all circumstances.
5. `pinned`: Packages pinned to a specific version, either via `pinned_packages` in your `.condarc` or defined in a `$PREFIX/conda-meta/pinned` file.
6. `virtual`: System properties exposed as virtual packages (e.g. `__glibc=2.17`). They can't really be installed or uninstalled, but they do participate in the solver by adding runtime constraints.
7. `do_not_remove`: A fixed list of packages that receive special treatment by the solver due to poor metadata in the early days of conda packaging. A legacy leftover.

This one group *does* change during the solver lifetime:

8. `conflicting`: Specs that are suspected to be a conflict for the solver.

Also, two more sources that are not obvious at first. These are not labeled as a *source*, but they do participate in the `specs` collection:

- In new environments, packages included in the `contex.create_default_packages` list. These `MatchSpec` objects are injected in each `conda create` command, so the solver will see them as explicitly requested by the user (`requested`).
- Specs added by command line modifiers. The specs here present aren't new (they are already in other categories), but they might end up in the `specs` list only when a flag is added. For example, `update --all` will add all the installed packages to the `specs` list, with no version constraint. Without this flag, the installed packages will still end up in the `specs` list, but with full constraints (`--freeze-installed` defaults for the first attempt) unless:
  - Frozen attempt failed.
  - `--update-specs` (or any other `UpdateModifier`) was passed, overriding `--freeze-installed`.

See? It gets involved. We will also use this vocabulary to help narrow down the type of change being done:

Types of `spec` objects:

- `specs`: map of package name to its currently corresponding `MatchSpec` instance.
- `spec`: specific instance of a `MatchSpec` object.
- Exact or frozen `spec`: a `spec` where both the `version` and `build` fields are constrained with == operators (exact match).

- Fully constrained or tight `spec`: a `spec` where both `version` and `build` are populated, but not necessarily with equality operators. It can also be inequalities (>, <, etc.) and fuzzy matches (`*something*`).
- Version-only `spec`: a `spec` where *only* the `version` field is populated. The `build` is not.
- Name-only, bare, or unconstrained `spec`: a `spec` with no `version` or `build` fields. Just the name of the package.
- Targeted `spec`: a `spec` with the `target` field populated. Extracted from the comments in the solver logic:

  > `target` is a reference to the package currently existing in the environment. Setting `target` instructs the solver to not disturb that package if it's not necessary. If the spec.name is being modified by inclusion in `specs_to_add`, we don't set `target`, since we *want* the solver to modify/update that package.
  >
  > TL;DR: when working with `MatchSpec` objects,
  >
  > - to minimize the version change, set `MatchSpec(name=name, target=prec.dist_str())`
  > - to freeze the package, set all the components of `MatchSpec` individually

- if the `spec` object does not have an adjective, it should be assumed it's being added to the `specs` map unmodified, as it came from its origin.

Pools (collections of `PackageRecord` objects):

- Installed pool: The installed packages, grouped by name. Each group should only contain one record!
- Explicit pool: The full index, but reduced for the specs in `requested`.

The following sections will get dry and to the point. They will state what output to expect from a given set of initial conditions. At least we'll try. Take into account that the `specs` list is kept around across attempts! In other words, the `specs` list is only really empty in the first attempt; if this fails, the subsequent attempts will only overwrite (update) the existing one. In practice, this should only affect how constrained packages are. The names should be the same.

It will also depend on whether we are adding (`conda install|create|update`) or removing (`conda remove`) packages. There's a common initialization part for both, but after that the logic is separate.

### Common initialization

> Note: This happens in `Solver._collect_all_metadata()`

This happens regardless of the type of command we are using (`install`, `update`, `create` or `remove`).

1. Add specs from `history`, if any.
2. Add specs from `do_not_remove`, but only if:
   - There's no spec for that name in `specs` already, *and*
   - A package with that name is *not* installed.
3. Add `virtual` packages as unconstrained `specs`.
4. Add all those installed packages, as unconstrained `specs`, that satisfy any of these conditions:
   - The history is empty (in that case, all installed packages are added)
   - The package name is part of `aggresive_updates`
   - The package was not installed by `conda`, but by pip or other PyPI tools instead.

**Preparing the index**

At this point, the populated `specs` and the `requested` specs are merged together. This temporary collection is used to determine how to reduce the index.

#### Processing specs for `conda install`

#### Preparation

1. Generate the explicit pool for the requested specs (via `Resolve._get_package_pool()`).
2. Detect potential conflicts (via (`Resolve.get_conflicting_specs()`).

#### Refine `specs` that match installed records

1. Check that each of `specs` match a single installed package or none! If there are two or more matches, it means that the environment is in bad shape and is basically broken. If the `spec` matches one installed package (let's call it *installed match*), we will modify the original `spec`.
2. We will turn the `spec` into an *exact* (frozen) spec if:
    1. The installed match is unmanageable (installed by pip, virtual, etc.)
    2. There's no history, we are not in `--freeze-installed` mode, and:
        - The spec is not a potential conflict, *and*
        - The package name *cannot* be found in the explicit pool index or, if it is, the installed match can be found in that explicit pool (to guarantee it will be found instead of creating one more conflict *just because*).
3. We relax the `spec` to a name-only `spec` if it's part of the aggressive updates list.
4. We turn it into a targeted `spec` if:
    1. The spec is in `history`. In that case, we take its *historic* spec counterpart and set the target to the installed match version and build.
    2. None of the above conditions were met. In other words, we'll try our best to match the installed package if none of the above applies, but if we fail we'll stick to whatever was already present in the `specs`.

#### Handle pinned specs

#### Processing specs for `conda remove`

### 4.6.10 API

As of conda 4.4, conda can be installed in any environment, not just environments with names starting with _ (underscore). That change was made, in part, so that conda can be used as a Python library.

As of conda 4.5, we do not support `pip install conda`. However, we are considering that as a supported bootstrap method in the future.

### conda

OS-agnostic, system-level binary package manager.

### __main__

Conda as a module entry point.

### _vendor

### _version

**version:** str

**__version__:** str

**__version_tuple__:** VERSION_TUPLE

**version_tuple:** VERSION_TUPLE

### activate

Conda activate and deactivate logic.

Implementation for all shell interface logic exposed via *conda shell.* [activate|deactivate|reactivate|hook|commands]*. This includes a custom argument parser, an abstract shell class, and special path handling for Windows.

See conda.cli.main.main_sourced for the entry point into this module.

## Classes

| | |
|---|---|
| *_Activator* | |
| *PosixActivator* | |
| *CshActivator* | |
| *XonshActivator* | |
| *CmdExeActivator* | |
| *FishActivator* | |
| *PowerShellActivator* | |
| *JSONFormatMixin* | Returns the necessary values for activation as JSON, so that tools can use them. |

### Functions

| | |
|---|---|
| *expand*(path) | |
| *ensure_binary*(value) | |
| *ensure_fs_path_encoding*(value) | |
| *backslash_to_forwardslash*(→ str \| tuple[str, ...) | |
| *_build_activator_cls*(shell) | Dynamically construct the activator class. |

### Attributes

| |
|---|
| *BUILTIN_COMMANDS* |
| *activator_map* |
| *formatter_map* |

**BUILTIN_COMMANDS**

**class _Activator**(*arguments=None*)

**pathsep_join:** str

**sep:** str

**path_conversion:** collections.abc.Callable[[str | collections.abc.Iterable[str] | None], str | tuple[str, Ellipsis] | None]

**script_extension:** str

**tempfile_extension:** str | None

**command_join:** str

**unset_var_tmpl:** str

**export_var_tmpl:** str

**path_var_tmpl:** str

**set_var_tmpl:** str

**run_script_tmpl:** str

**hook_source_path:** pathlib.Path | None

**inline_hook_source:** bool

**get_export_unset_vars**(*export_metavars=True*, ***kwargs*)

**Parameters**

- **export_metavars** -- whether to export *conda_exe_vars* meta variables.
- **kwargs** -- environment variables to export. .. if you pass and set any other variable to None, then it emits it to the dict with a value of None.

**Returns**
A dict of env vars to export ordered the same way as kwargs. And a list of env vars to unset.

**_finalize**(*commands*, *ext*)

**activate**()

**deactivate**()

**reactivate**()

**hook**(*auto_activate: bool | None = None*) → str

**execute**()

**commands**()
Returns a list of possible subcommands that are valid immediately following *conda* at the command line. This method is generally only used by tab-completion.

**template_unset_var**(*key: str*) → str

**template_export_var**(*key: str*, *value: str*) → str

**template_path_var**(*key: str*, *value: str*) → str

**_hook_preamble**() → str | None

**_hook_postamble**() → str | None

**_parse_and_set_args**() → None

**_yield_commands**(*cmds_dict*)

**build_activate**(*env_name_or_prefix*)

**build_stack**(*env_name_or_prefix*)

**_build_activate_stack**(*env_name_or_prefix*, *stack*)

**build_deactivate**()

**build_reactivate**()

**_get_starting_path_list**()

**_get_path_dirs**(*prefix*)

**_add_prefix_to_path**(*prefix*, *starting_path_dirs=None*)

**_remove_prefix_from_path**(*prefix*, *starting_path_dirs=None*)

**_replace_prefix_in_path**(*old_prefix*, *new_prefix*, *starting_path_dirs=None*)

**_update_prompt**(*set_vars*, *conda_prompt_modifier*)

**_default_env**(*prefix*)

**_prompt_modifier**(*prefix*, *conda_default_env*)

**_get_activate_scripts**(*prefix*)

**_get_deactivate_scripts**(*prefix*)

**_get_environment_env_vars**(*prefix*)

**expand**(*path*)

**ensure_binary**(*value*)

**ensure_fs_path_encoding**(*value*)

**backslash_to_forwardslash**(*paths: str | collections.abc.Iterable[str] | None*) → str | tuple[str, Ellipsis] | None

**class PosixActivator**(*arguments=None*)

Bases: *_Activator*

**pathsep_join**

**sep = '/'**

**path_conversion**

**script_extension = '.sh'**

**tempfile_extension**

**command_join = '\n'**

**unset_var_tmpl = "export %s=''"**

**export_var_tmpl = "export %s='%s'"**

**path_var_tmpl**

**set_var_tmpl = "%s='%s'"**

**run_script_tmpl**

**hook_source_path**

**inline_hook_source = True**

**_update_prompt**(*set_vars*, *conda_prompt_modifier*)

**class CshActivator**(*arguments=None*)

Bases: *_Activator*

**pathsep_join**

**sep = '/'**

**path_conversion**

**script_extension = '.csh'**

**tempfile_extension**

**command_join = ';\n'**

**unset_var_tmpl = 'unsetenv %s'**

**export_var_tmpl = 'setenv %s "%s"'**

**path_var_tmpl**

**set_var_tmpl = "set %s='%s'"**

**run_script_tmpl**

**hook_source_path**

**inline_hook_source = False**

**_update_prompt**(*set_vars*, *conda_prompt_modifier*)

**class XonshActivator**(*arguments=None*)

Bases: *_Activator*

**pathsep_join**

**sep = '/'**

**path_conversion**

**script_extension**

**tempfile_extension**

**command_join = '\n'**

**unset_var_tmpl = Multiline-String**

```
"""try:
    del $%s
except KeyError:
    pass"""
```

**export_var_tmpl = "$%s = '%s'"**

**path_var_tmpl**

**set_var_tmpl**

**run_script_tmpl**

**hook_source_path**

**inline_hook_source = True**

**template_path_var**(*key: str*, *value: str*) → str

**class CmdExeActivator**(*arguments=None*)

Bases: *_Activator*

**pathsep_join**

sep = '\\'

path_conversion

script_extension = '.bat'

tempfile_extension = '.env'

command_join = '\n'

unset_var_tmpl = '%s='

export_var_tmpl = '%s=%s'

path_var_tmpl

set_var_tmpl

run_script_tmpl = '_CONDA_SCRIPT=%s'

hook_source_path

inline_hook_source

_update_prompt(*set_vars*, *conda_prompt_modifier*)

_hook_preamble() → None

class FishActivator(*arguments=None*)

Bases: *_Activator*

pathsep_join

sep = '/'

path_conversion

script_extension = '.fish'

tempfile_extension

command_join = ';\n'

unset_var_tmpl = 'set -e %s'

export_var_tmpl = 'set -gx %s "%s"'

path_var_tmpl

set_var_tmpl = 'set -g %s "%s"'

run_script_tmpl = 'source "%s"'

hook_source_path

inline_hook_source = True

class PowerShellActivator(*arguments=None*)

Bases: *_Activator*

pathsep_join

**sep**

**path_conversion**

**script_extension = '.ps1'**

**tempfile_extension**

**command_join = '\n'**

**unset_var_tmpl = '$Env:%s = $null'**

**export_var_tmpl = '$Env:%s = "%s"'**

**path_var_tmpl**

**set_var_tmpl**

**run_script_tmpl = '. "%s"'**

**hook_source_path**

**inline_hook_source = True**

**_hook_preamble**() → str

**_hook_postamble**() → str

**class JSONFormatMixin**(*arguments=None*)

Bases: *_Activator*

Returns the necessary values for activation as JSON, so that tools can use them.

**pathsep_join**

**tempfile_extension**

**command_join**

**_hook_preamble**()

**_finalize**(*commands*, *ext*)

**_yield_commands**(*cmds_dict*)

**activator_map: dict[str, type[_Activator]]**

**formatter_map**

**_build_activator_cls**(*shell*)

Dynamically construct the activator class.

Detect the base activator and any number of formatters (appended using '+' to the base name). For example, *posix+json* (as in *conda shell.posix+json activate*) would use the *PosixActivator* base class and add the *JSONFormatMixin*.

### api

Collection of conda's high-level APIs.

## Classes

| | |
|---|---|
| *Solver* | **Beta** While in beta, expect both major and minor changes across minor releases. |
| *SubdirData* | **Beta** While in beta, expect both major and minor changes across minor releases. |
| *PackageCacheData* | **Beta** While in beta, expect both major and minor changes across minor releases. |
| *PrefixData* | **Beta** While in beta, expect both major and minor changes across minor releases. |

## Attributes

| |
|---|
| *DepsModifier* |
| *UpdateModifier* |

**DepsModifier**

**UpdateModifier**

**class Solver**(*prefix*, *channels*, *subdirs=()*, *specs_to_add=()*, *specs_to_remove=()*)

**Beta** While in beta, expect both major and minor changes across minor releases.

A high-level API to conda's solving logic. Three public methods are provided to access a solution in various forms.

- *solve_final_state()*
- *solve_for_diff()*
- *solve_for_transaction()*

**Beta**

**Parameters**

- **prefix** (*str*) -- The conda prefix / environment location for which the *Solver* is being instantiated.
- **channels** (Sequence[Channel]) -- A prioritized list of channels to use for the solution.
- **subdirs** (*Sequence[str]*) -- A prioritized list of subdirs to use for the solution.
- **specs_to_add** (set[MatchSpec]) -- The set of package specs to add to the prefix.
- **specs_to_remove** (set[MatchSpec]) -- The set of package specs to remove from the prefix.

**solve_final_state**(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Gives the final, solved state of the environment.

**Parameters**

- **deps_modifier** (DepsModifier) -- An optional flag indicating special solver handling for dependencies. The default solver behavior is to be as conservative as possible with dependency updates (in the case the dependency already exists in the environment), while still ensuring all dependencies are satisfied. Options include * NO_DEPS * ONLY_DEPS * UPDATE_DEPS * UPDATE_DEPS_ONLY_DEPS * FREEZE_INSTALLED
- **prune** (*bool*) -- If `True`, the solution will not contain packages that were previously brought into the environment as dependencies but are no longer required as dependencies and are not user-requested.
- **ignore_pinned** (*bool*) -- If `True`, the solution will ignore pinned package configuration for the prefix.
- **force_remove** (*bool*) -- Forces removal of a package without removing packages that depend on it.

**Returns**

In sorted dependency order from roots to leaves, the package references for the solved state of the environment.

**Return type**

tuple[PackageRef]

**solve_for_diff**(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=False*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Gives the package references to remove from an environment, followed by the package references to add to an environment.

**Parameters**

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- For requested specs_to_add that are already satisfied in the environment, instructs the solver to remove the package and spec from the environment, and then add it back--possibly with the exact package instance modified, depending on the spec exactness.

**Returns**

A two-tuple of PackageRef sequences. The first is the group of packages to remove from the environment, in sorted dependency order from leaves to roots. The second is the group of packages to add to the environment, in sorted dependency order from roots to leaves.

**Return type**

tuple[PackageRef], tuple[PackageRef]

`solve_for_transaction`(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=False*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Gives an UnlinkLinkTransaction instance that can be used to execute the solution on an environment.

**Parameters**

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- See *solve_for_diff()*.

**Return type**
*UnlinkLinkTransaction*

`class SubdirData`(*channel*)

**Beta** While in beta, expect both major and minor changes across minor releases.

High-level management and usage of repodata.json for subdirs.

**Beta** While in beta, expect both major and minor changes across minor releases.

**Parameters**
**channel** (*str or Channel*) -- The target subdir for the instance. Must either be a url that includes a subdir or a Channel that includes a subdir. e.g.:

- 'https://repo.anaconda.com/pkgs/main/linux-64'
- Channel('https://repo.anaconda.com/pkgs/main/linux-64')
- Channel('conda-forge/osx-64')

`query`(*package_ref_or_match_spec*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Run a query against this specific instance of repodata.

**Parameters**
**package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

**Returns**
tuple[PackageRecord]

`static query_all`(*package_ref_or_match_spec*, *channels=None*, *subdirs=None*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Run a query against all repodata instances in channel/subdir matrix.

**Parameters**

- **package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.
- **channels** (*Iterable[Channel or str] or None*) -- An iterable of urls for channels or Channel objects. If None, will fall back to context.channels.

- **subdirs** (*Iterable[str] or None*) -- If None, will fall back to context.subdirs.

**Returns**
tuple[PackageRecord]

`iter_records()`

**Beta** While in beta, expect both major and minor changes across minor releases.

**Returns**

**A generator over all records contained in the repodata.json**
instance. Warning: this is a generator that is exhausted on first use.

**Return type**
Iterable[*PackageRecord*]

`reload()`

**Beta** While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. repodata.json) is lazily downloaded/loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

**Returns**
SubdirData

`class PackageCacheData(pkgs_dir)`

**Beta** While in beta, expect both major and minor changes across minor releases.

High-level management and usage of package caches.

**Beta** While in beta, expect both major and minor changes across minor releases.

**Parameters**
**pkgs_dir** (*str*)

`property is_writable`

**Beta** While in beta, expect both major and minor changes across minor releases.

Indicates if the package cache location is writable or read-only.

**Returns**
bool

`get(package_ref, default=NULL)`

**Beta** While in beta, expect both major and minor changes across minor releases.

**Parameters**

- **package_ref** (*PackageRef*) -- A `PackageRef` instance representing the key for the `PackageCacheRecord` being sought.
- **default** -- The default value to return if the record does not exist. If not specified and no record exists, `KeyError` is raised.

**Returns**
PackageCacheRecord

`query(package_ref_or_match_spec)`

**Beta** While in beta, expect both major and minor changes across minor releases.

Run a query against this specific package cache instance.

**Parameters**
**package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.

**Returns**
tuple[PackageCacheRecord]

**static query_all**(*package_ref_or_match_spec*, *pkgs_dirs=None*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Run a query against all package caches.

**Parameters**

- **package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact PackageRef to match against, or a MatchSpec query object. A str will be turned into a MatchSpec automatically.
- **pkgs_dirs** (*Iterable[str] or None*) -- If None, will fall back to context.pkgs_dirs.

**Returns**
tuple[PackageCacheRecord]

**iter_records**()

**Beta** While in beta, expect both major and minor changes across minor releases.

**Returns**

**A generator over all records contained in the package**
cache instance. Warning: this is a generator that is exhausted on first use.

**Return type**
Iterable[*PackageCacheRecord*]

**static first_writable**(*pkgs_dirs=None*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Get an instance object for the first writable package cache.

**Parameters**
**pkgs_dirs** (*Iterable[str]*) -- If None, will fall back to context.pkgs_dirs.

**Returns**
An instance for the first writable package cache.

**Return type**
*PackageCacheData*

**reload**()

**Beta** While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. contents of the pkgs_dir) is lazily loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

**Returns**
PackageCacheData

**class PrefixData**(*prefix_path*)

**Beta** While in beta, expect both major and minor changes across minor releases.

High-level management and usage of conda environment prefixes.

**Beta** While in beta, expect both major and minor changes across minor releases.

**Parameters**
**prefix_path** (*str*)

`property is_writable`

**Beta** While in beta, expect both major and minor changes across minor releases.

Indicates if the prefix is writable or read-only.

**Returns**
True if the prefix is writable. False if read-only. None if the prefix does not exist as a conda environment.

**Return type**
bool or None

`get`(*package_ref*, *default=NULL*)

**Beta** While in beta, expect both major and minor changes across minor releases.

**Parameters**

- **package_ref** (*PackageRef*) -- A `PackageRef` instance representing the key for the `PrefixRecord` being sought.
- **default** -- The default value to return if the record does not exist. If not specified and no record exists, `KeyError` is raised.

**Returns**
PrefixRecord

`query`(*package_ref_or_match_spec*)

**Beta** While in beta, expect both major and minor changes across minor releases.

Run a query against this specific prefix instance.

**Parameters**
**package_ref_or_match_spec** (*PackageRef or MatchSpec or str*) -- Either an exact `PackageRef` to match against, or a `MatchSpec` query object. A `str` will be turned into a `MatchSpec` automatically.

**Returns**
tuple[PrefixRecord]

`iter_records`()

**Beta** While in beta, expect both major and minor changes across minor releases.

**Returns**

**A generator over all records contained in the prefix.**
Warning: this is a generator that is exhausted on first use.

**Return type**
Iterable[*PrefixRecord*]

`reload`()

**Beta** While in beta, expect both major and minor changes across minor releases.

Update the instance with new information. Backing information (i.e. contents of the conda-meta directory) is lazily loaded on first use by the other methods of this class. You should only use this method if you are *sure* you have outdated data.

**Returns**
PrefixData

### auxlib

Auxlib is an auxiliary library to the python standard library.

The aim is to provide core generic features for app development in python. Auxlib fills in some python stdlib gaps much like pytoolz has for functional programming, pyrsistent has for data structures, or boltons has generally.

**Major areas addressed include:**

- packaging: package versioning, with a clean and less invasive alternative to versioneer
- entity: robust base class for type-enforced data models and transfer objects
- type_coercion: intelligent type coercion utilities
- *Configuration*: a map implementation designed specifically to hold application configuration and context information
- factory: factory pattern implementation
- path: file path utilities especially helpful when working with various python package formats
- logz: logging initialization routines to simplify python logging setup
- crypt: simple, but correct, pycrypto wrapper

[2021-11-09] Our version of auxlib has deviated from the upstream project by a significant amount (especially compared with the other vendored packages). Further, the upstream project has low popularity and is no longer actively maintained. Consequently it was decided to absorb, refactor, and replace auxlib. As a first step of this process we moved conda._vendor.auxlib to conda.auxlib.

### collection

Common collection classes.

#### Classes

| | |
|---|---|
| *AttrDict* | Sub-classes dict, and further allows attribute-like access to dictionary items. |

#### Functions

| | |
|---|---|
| *first*(seq[, key, default, apply]) | Give the first value that satisfies the key test. |
| *last*(seq[, key, default, apply]) | |

**class AttrDict**(**args*, ***kwargs*)

Bases: dict

Sub-classes dict, and further allows attribute-like access to dictionary items.

### Examples

```
>>> d = AttrDict({'a': 1})
>>> d.a, d['a'], d.get('a')
(1, 1, 1)
>>> d.b = 2
>>> d.b, d['b']
(2, 2)
```

Initialize self. See help(type(self)) for accurate signature.

**first**(*seq*, *key=bool*, *default=None*, *apply=lambda x: ...*)

Give the first value that satisfies the key test.

**Parameters**

- **seq** (*iterable*)
- **key** (*callable*) -- test for each element of iterable
- **default** -- returned when all elements fail test
- **apply** (*callable*) -- applied to element before return, but not to default value

Returns: first element in seq that passes key, mutated with optional apply

### Examples

```
>>> first([0, False, None, [], (), 42])
42
>>> first([0, False, None, [], ()]) is None
True
>>> first([0, False, None, [], ()], default='ohai')
'ohai'
>>> import re
>>> m = first(re.match(regex, 'abc') for regex in ['b.*', 'a(.*)'])
>>> m.group(1)
'bc'
```

The optional *key* argument specifies a one-argument predicate function like that used for *filter()*. The *key* argument, if supplied, must be in keyword form. For example: >>> first([1, 1, 3, 4, 5], key=lambda x: x % 2 == 0) 4

**last**(*seq*, *key=bool*, *default=None*, *apply=lambda x: ...*)

## compat

### Functions

| | |
|---|---|
| *isiterable*(obj) | |
| *shlex_split_unicode*(to_split[, posix]) | |
| *Utf8NamedTemporaryFile*([mode, buffering, newline, ...]) | |

**isiterable**(*obj*)

**shlex_split_unicode**(*to_split*, *posix=True*)

**Utf8NamedTemporaryFile**(*mode='w+b'*, *buffering=-1*, *newline=None*, *suffix=None*, *prefix=None*, *dir=None*, *delete=True*)

## decorators

### Classes

| | |
|---|---|
| *classproperty* | |

### Functions

| | |
|---|---|
| *memoizemethod*(method) | Decorator to cause a method to cache it's results in self for each |
| *clear_memoized_methods*(obj, *method_names) | Clear the memoized method or @memoizedproperty results for the given |
| *memoizedproperty*(func) | Decorator to cause a method to cache it's results in self for each |

**memoizemethod**(*method*)

Decorator to cause a method to cache it's results in self for each combination of inputs and return the cached result on subsequent calls. Does not support named arguments or arg values that are not hashable.

```
>>> class Foo (object):
...   @memoizemethod
...   def foo(self, x, y=0):
...     print('running method with', x, y)
...     return x + y + 3
...
>>> foo1 = Foo()
>>> foo2 = Foo()
>>> foo1.foo(10)
running method with 10 0
```

(continues on next page)

(continued from previous page)

```
13
>>> foo1.foo(10)
13
>>> foo2.foo(11, y=7)
running method with 11 7
21
>>> foo2.foo(11)
running method with 11 0
14
>>> foo2.foo(11, y=7)
21
>>> class Foo (object):
...   def __init__(self, lower):
...     self.lower = lower
...   @memoizemethod
...   def range_tuple(self, upper):
...     print('running function')
...     return tuple(i for i in range(self.lower, upper))
...   @memoizemethod
...   def range_iter(self, upper):
...     print('running function')
...     return (i for i in range(self.lower, upper))
...
>>> foo = Foo(3)
>>> foo.range_tuple(6)
running function
(3, 4, 5)
>>> foo.range_tuple(7)
running function
(3, 4, 5, 6)
>>> foo.range_tuple(6)
(3, 4, 5)
>>> foo.range_iter(6)
Traceback (most recent call last):
TypeError: Can't memoize a generator or non-hashable object!
```

**clear_memoized_methods**(*obj*, **method_names*)

Clear the memoized method or @memoizedproperty results for the given method names from the given object.

```
>>> v = [0]
>>> def inc():
...     v[0] += 1
...     return v[0]
...
>>> class Foo(object):
...    @memoizemethod
...    def foo(self):
...        return inc()
...    @memoizedproperty
...    def g(self):
...       return inc()
...
```

(continues on next page)

(continued from previous page)

```
>>> f = Foo()
>>> f.foo(), f.foo()
(1, 1)
>>> clear_memoized_methods(f, 'foo')
>>> (f.foo(), f.foo(), f.g, f.g)
(2, 2, 3, 3)
>>> (f.foo(), f.foo(), f.g, f.g)
(2, 2, 3, 3)
>>> clear_memoized_methods(f, 'g', 'no_problem_if_undefined')
>>> f.g, f.foo(), f.g
(4, 2, 4)
>>> f.foo()
2
```

**memoizedproperty**(*func*)

Decorator to cause a method to cache it's results in self for each combination of inputs and return the cached result on subsequent calls. Does not support named arguments or arg values that are not hashable.

```
>>> class Foo (object):
...   _x = 1
...   @memoizedproperty
...   def foo(self):
...     self._x += 1
...     print('updating and returning {0}'.format(self._x))
...     return self._x
...
>>> foo1 = Foo()
>>> foo2 = Foo()
>>> foo1.foo
updating and returning 2
2
>>> foo1.foo
2
>>> foo2.foo
updating and returning 2
2
>>> foo1.foo
2
```

**class classproperty**(*getter=None*, *setter=None*)

**__get__**(*obj*, *type_=None*)

**__set__**(*obj*, *value*)

**setter**(*setter*)

### entity

This module provides serializable, validatable, type-enforcing domain objects and data transfer objects. It has many of the same motivations as the python Marshmallow package. It is most similar to Schematics.

#### Tutorial

#### Chapter 1: Entity and Field Basics

```
>>> class Color(Enum):
...     blue = 0
...     black = 1
...     red = 2
>>> class Car(Entity):
...     weight = NumberField(required=False)
...     wheels = IntField(default=4, validation=lambda x: 3 <= x <= 4)
...     color = EnumField(Color)
```

```
>>> # create a new car object
>>> car = Car(color=Color.blue, weight=4242.46)
>>> car
Car(weight=4242.46, color=0)
```

```
>>> # it has 4 wheels, all by default
>>> car.wheels
4
```

```
>>> # but a car can't have 5 wheels!
>>> #  the `validation=` field is a simple callable that returns a
>>> #  boolean based on validity
>>> car.wheels = 5
Traceback (most recent call last):
ValidationError: Invalid value 5 for wheels
```

```
>>> # we can call .dump() on car, and just get back a standard
>>> #  python dict actually, it's an ordereddict to match attribute
>>> #  declaration order
>>> type(car.dump())
<class '...OrderedDict'>
>>> car.dump()
OrderedDict([('weight', 4242.46), ('wheels', 4), ('color', 0)])
```

```
>>> # and json too (note the order!)
>>> car.json()
'{"weight": 4242.46, "wheels": 4, "color": 0}'
```

```
>>> # green cars aren't allowed
>>> car.color = "green"
Traceback (most recent call last):
ValidationError: 'green' is not a valid Color
```

```
>>> # but black cars are!
>>> car.color = "black"
>>> car.color
<Color.black: 1>
```

```
>>> # car.color really is an enum, promise
>>> type(car.color)
<enum 'Color'>
```

```
>>> # enum assignment can be with any of (and preferentially)
>>> #   (1) an enum literal,
>>> #   (2) a valid enum value, or
>>> #   (3) a valid enum name
>>> car.color = Color.blue; car.color.value
0
>>> car.color = 1; car.color.name
'black'
```

```
>>> # let's do a round-trip marshalling of this thing
>>> same_car = Car.from_json(car.json())  # or equally Car.from_json(json.dumps(car.
↪dump()))
>>> same_car == car
True
```

```
>>> # actually, they're two different instances
>>> same_car is not car
True
```

```
>>> # this works too
>>> cloned_car = Car(**car.dump())
>>> cloned_car == car
True
```

```
>>> # while we're at it, these are all equivalent too
>>> car == Car.from_objects(car)
True
>>> car == Car.from_objects({"weight": 4242.46, "wheels": 4, "color": 1})
True
>>> car == Car.from_json('{"weight": 4242.46, "color": 1}')
True
```

```
>>> # .from_objects() even lets you stack and combine objects
>>> class DumbClass:
...     color = 0
...     wheels = 3
>>> Car.from_objects(DumbClass(), dict(weight=2222, color=1))
Car(weight=2222, wheels=3, color=0)
>>> # and also pass kwargs that override properties pulled
>>> #  off any objects
>>> Car.from_objects(DumbClass(), {'weight': 2222, 'color': 1}, color=2, weight=33)
Car(weight=33, wheels=3, color=2)
```

### Chapter 2: Entity and Field Composition

```
>>> # now let's get fancy
>>> # a ComposableField "nests" another valid Entity
>>> # a ListField's first argument is a "generic" type,
>>> #   which can be a valid Entity, any python primitive
>>> #   type, or a list of Entities/types
>>> class Fleet(Entity):
...     boss_car = ComposableField(Car)
...     cars = ListField(Car)
```

```
>>> # here's our fleet of company cars
>>> company_fleet = Fleet(boss_car=Car(color='red'), cars=[car, same_car, cloned_car])
>>> company_fleet.pretty_json()
{
  "boss_car": {
    "wheels": 4
    "color": 2,
  },
  "cars": [
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    },
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    },
    {
      "weight": 4242.46,
      "wheels": 4
      "color": 1,
    }
  ]
}
```

```
>>> # the boss' car is red of course (and it's still an Enum)
>>> company_fleet.boss_car.color.name
'red'
```

```
>>> # and there are three cars left for the employees
>>> len(company_fleet.cars)
3
```

### Chapter 3: Immutability

```
>>> class ImmutableCar(ImmutableEntity):
...     wheels = IntField(default=4, validation=lambda x: 3 <= x <= 4)
...     color = EnumField(Color)
>>> icar = ImmutableCar.from_objects({'wheels': 3, 'color': 'blue'})
>>> icar
ImmutableCar(wheels=3, color=0)
```

```
>>> icar.wheels = 4
Traceback (most recent call last):
AttributeError: Assignment not allowed. ImmutableCar is immutable.
```

```
>>> class FixedWheelCar(Entity):
...     wheels = IntField(default=4, immutable=True)
...     color = EnumField(Color)
>>> fwcar = FixedWheelCar.from_objects(icar)
>>> fwcar.json()
'{"wheels": 3, "color": 0}'
```

```
>>> # repainting the car is easy
>>> fwcar.color = Color.red
>>> fwcar.color.name
'red'
```

```
>>> # can't really change the number of wheels though
>>> fwcar.wheels = 18
Traceback (most recent call last):
AttributeError: The wheels field is immutable.
```

### Chapter X: The del and null Weeds

```
>>> old_date = lambda: isoparse('1982-02-17')
>>> class CarBattery(Entity):
...     # NOTE: default value can be a callable!
...     first_charge = DateField(required=False)  # default=None, nullable=False
...     latest_charge = DateField(default=old_date, nullable=True)  # required=True
...     expiration = DateField(default=old_date, required=False, nullable=False)
```

```
>>> # starting point
>>> battery = CarBattery()
>>> battery
CarBattery()
>>> battery.json()
'{"latest_charge": "1982-02-17T00:00:00", "expiration": "1982-02-17T00:00:00"}'
```

```
>>> # first_charge is not assigned a default value. Once one is assigned, it can be␣
↪deleted,
>>> #   but it can't be made null.
```

(continues on next page)

(continued from previous page)

```
>>> battery.first_charge = isoparse('2016-03-23')
>>> battery
CarBattery(first_charge=datetime.datetime(2016, 3, 23, 0, 0))
>>> battery.first_charge = None
Traceback (most recent call last):
ValidationError: Value for first_charge not given or invalid.
>>> del battery.first_charge
>>> battery
CarBattery()
```

```
>>> # latest_charge can be null, but it can't be deleted. The default value is a callable.
>>> del battery.latest_charge
Traceback (most recent call last):
AttributeError: The latest_charge field is required and cannot be deleted.
>>> battery.latest_charge = None
>>> battery.json()
'{"latest_charge": null, "expiration": "1982-02-17T00:00:00"}'
```

```
>>> # expiration is assigned by default, can't be made null, but can be deleted.
>>> battery.expiration
datetime.datetime(1982, 2, 17, 0, 0)
>>> battery.expiration = None
Traceback (most recent call last):
ValidationError: Value for expiration not given or invalid.
>>> del battery.expiration
>>> battery.json()
'{"latest_charge": null}'
```

### Classes

| | |
|---|---|
| *Field* | Fields are doing something very similar to boxing and unboxing |
| *BooleanField* | Fields are doing something very similar to boxing and unboxing |
| *IntegerField* | Fields are doing something very similar to boxing and unboxing |
| *NumberField* | Fields are doing something very similar to boxing and unboxing |
| *StringField* | Fields are doing something very similar to boxing and unboxing |
| *DateField* | Fields are doing something very similar to boxing and unboxing |
| *EnumField* | Fields are doing something very similar to boxing and unboxing |
| *ListField* | Fields are doing something very similar to boxing and unboxing |
| *MapField* | Fields are doing something very similar to boxing and unboxing |
| *ComposableField* | Fields are doing something very similar to boxing and unboxing |
| *Entity* | |
| *ImmutableEntity* | |

### Attributes

| | |
|---|---|
| *BoolField* | |
| *IntField* | |

**class Field**(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

property name

property required

property type

property default

property in_dump

property default_in_dump

property nullable

property is_nullable

property immutable

_order_helper = 0

set_name(*name*)

__get__(*instance*, *instance_type*)

__set__(*instance*, *val*)

__delete__(*instance*)

box(*instance*, *instance_type*, *val*)

unbox(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

validate(*instance*, *val*)

**Returns**
if val is valid

**Return type**
True

**Raises**
*ValidationError* --

class BooleanField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)

- **dump** (*boolean, optional*)

_type

box(*instance*, *instance_type*, *val*)

BoolField

class IntegerField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

_type

IntField

class NumberField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

_type = ()

class StringField(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**_type**

**box**(*instance*, *instance_type*, *val*)

class **DateField**(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**_type**

**box**(*instance*, *instance_type*, *val*)

**dump**(*instance*, *instance_type*, *val*)

class **EnumField**(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**box**(*instance*, *instance_type*, *val*)

**dump**(*instance*, *instance_type*, *val*)

**class ListField**(*element_type*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**_type**

**box**(*instance*, *instance_type*, *val*)

**unbox**(*instance*, *instance_type*, *val*)

**dump**(*instance*, *instance_type*, *val*)

**validate**(*instance*, *val*)

**Returns**
if val is valid

**Return type**
True

**Raises**
*ValidationError* --

**class MapField**(*default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=True*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

_type

box(*instance*, *instance_type*, *val*)

class ComposableField(*field_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *Field*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class Entity(**kwargs*)

property _initd

__fields__

_lazy_validate = False

classmethod from_objects(**objects*, ***override_fields*)

Construct a new object of type `cls` from existing objects or dicts.

Allows the creation of new objects of concrete *Entity* subclasses by combining information from several sources. This can be any combination of objects and dictionaries passed in as positional arguments. When looking for the value of the fields of the *Entity* subclass, the first object that provides an attribute (or, in the case of a dict an entry) that has the name of the field or one of its aliases will take precedence. Any keyword arguments passed in will override this and take precedence.

**Parameters**

- **cls** (*Entity* subclass) -- The class to create, usually determined by call, e.g. `PrefixRecord.from_objects(...)`.
- ***objects** (*tuple(object or dict)*) -- Any combination of objects and dicts in order of decending precedence.
- ****override_fields** (*dict(str, object)*) -- Any individual fields overriding possible contents from `*objects`.

classmethod from_json(*json_str*)

classmethod load(*data_dict*)

validate()

**__repr__**()
Return repr(self).

**classmethod __register__**()

**json**(*indent=None*, *separators=None*, ***kwargs*)

**pretty_json**(*indent=2*, *separators=(',', ': ')*, ***kwargs*)

**dump**()

**classmethod __dump_fields**()

**__eq__**(*other*)
Return self==value.

**__hash__**()
Return hash(self).

**class ImmutableEntity**(***kwargs*)
Bases: *Entity*

**__setattr__**(*attribute*, *value*)
Implement setattr(self, name, value).

**__delattr__**(*item*)
Implement delattr(self, name).

### exceptions

## Classes

| | |
|---|---|
| *AuxlibError* | Mixin to identify exceptions associated with the auxlib package. |

## Functions

| | |
|---|---|
| *Raise*(exception) | |

**Raise**(*exception*)

**class AuxlibError**
Mixin to identify exceptions associated with the auxlib package.

**exception ValidationError**(*key*, *value=None*, *valid_types=None*, *msg=None*)
Bases: *AuxlibError*, TypeError

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

**exception ThisShouldNeverHappenError**

Bases: *AuxlibError*, AttributeError

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

**ish**

### Functions

| | | | |
|---|---|---|---|
| *dals*(string) | | | dedent and left-strip |
| *_get_attr*(obj, attr_name[, aliases]) | | | |
| *find_or_none*(key, _map_index]) | search_maps[, | aliases, | Return the value of the first key found in the list of search_maps, |
| *find_or_raise*(key, _map_index]) | search_maps[, | aliases, | |

**dals**(*string*)

dedent and left-strip

**_get_attr**(*obj*, *attr_name*, *aliases=()*)

**find_or_none**(*key*, *search_maps*, *aliases=()*, *_map_index=0*)

Return the value of the first key found in the list of search_maps, otherwise return None.

#### Examples

```
>>> from .collection import AttrDict
>>> d1 = AttrDict({'a': 1, 'b': 2, 'c': 3, 'e': None})
>>> d2 = AttrDict({'b': 5, 'e': 6, 'f': 7})
>>> find_or_none('c', (d1, d2))
3
>>> find_or_none('f', (d1, d2))
7
>>> find_or_none('b', (d1, d2))
2
>>> print(find_or_none('g', (d1, d2)))
None
>>> find_or_none('e', (d1, d2))
6
```

**find_or_raise**(*key*, *search_maps*, *aliases=()*, *_map_index=0*)

### logz

### Functions

| | |
|---|---|
| *set_root_level*([level]) | |
| *attach_stderr*([level]) | |
| *detach_stderr*() | |
| *initialize_logging*([level]) | |
| *fullname*(obj) | |
| *request_header_sort_key*(item) | |
| *response_header_sort_key*(item) | |
| *stringify*(obj[, content_max_len]) | |

### Attributes

| | |
|---|---|
| *root_log* | |
| *NullHandler* | |
| *DEBUG_FORMATTER* | |
| *INFO_FORMATTER* | |
| *_DUMPS* | |
| *request_header_sort_dict* | |
| *response_header_sort_dict* | |

**root_log**

**NullHandler**

**DEBUG_FORMATTER**

**INFO_FORMATTER**

**set_root_level**(*level=INFO*)

**attach_stderr**(*level=INFO*)

**detach_stderr**()

**initialize_logging**(*level=INFO*)

**_DUMPS**

**fullname**(*obj*)

**request_header_sort_dict**

**request_header_sort_key**(*item*)

**response_header_sort_dict**

**response_header_sort_key**(*item*)

**stringify**(*obj*, *content_max_len=0*)

### type_coercion

Collection of functions to coerce conversion of types with an intelligent guess.

#### Functions

| | |
|---|---|
| *numberify*(value) | |
| *boolify*(value[, nullable, return_string]) | Convert a number, string, or sequence type into a pure boolean. |
| *typify*(value[, type_hint]) | Take a primitive value, usually a string, and try to make a more relevant type out of it. |
| *maybecall*(value) | |

**numberify**(*value*)

##### Examples

```
>>> [numberify(x) for x in ('1234', 1234, '0755', 0o0755, False, 0, '0', True, 1, '1
↪')]
  [1234, 1234, 755, 493, 0, 0, 0, 1, 1, 1]
>>> [numberify(x) for x in ('12.34', 12.34, 1.2+3.5j, '1.2+3.5j')]
[12.34, 12.34, (1.2+3.5j), (1.2+3.5j)]
```

**boolify**(*value*, *nullable=False*, *return_string=False*)

Convert a number, string, or sequence type into a pure boolean.

**Parameters**
**value** (*number, string, sequence*) -- pretty much anything

**Returns**
boolean representation of the given value

**Return type**
bool

### Examples

```
>>> [boolify(x) for x in ('yes', 'no')]
[True, False]
>>> [boolify(x) for x in (0.1, 0+0j, True, '0', '0.0', '0.1', '2')]
[True, False, True, False, False, True, True]
>>> [boolify(x) for x in ("true", "yes", "on", "y")]
[True, True, True, True]
>>> [boolify(x) for x in ("no", "non", "none", "off", "")]
[False, False, False, False, False]
>>> [boolify(x) for x in ([], set(), dict(), tuple())]
[False, False, False, False]
>>> [boolify(x) for x in ([1], set([False]), dict({'a': 1}), tuple([2]))]
[True, True, True, True]
```

**typify**(*value*, *type_hint=None*)

Take a primitive value, usually a string, and try to make a more relevant type out of it. An optional type_hint will try to coerce the value to that type.

**Parameters**

- **value** (*Any*) -- Usually a string, not a sequence
- **type_hint** (*type or tuple[type]*)

### Examples

```
>>> typify('32')
32
>>> typify('32', float)
32.0
>>> typify('32.0')
32.0
>>> typify('32.0.0')
'32.0.0'
>>> [typify(x) for x in ('true', 'yes', 'on')]
[True, True, True]
>>> [typify(x) for x in ('no', 'FALSe', 'off')]
[False, False, False]
>>> [typify(x) for x in ('none', 'None', None)]
[None, None, None]
```

**maybecall**(*value*)

**__version__ = '0.0.43'**

**__author__ = 'Kale Franz'**

**__email__ = 'kale@franz.io'**

```
__url__ = 'https://github.com/kalefranz/auxlib'

__license__ = 'ISC'

__copyright__ = '(c) 2015 Kale Franz. All rights reserved.'

__summary__ = 'auxiliary library to the python standard library'
```

### base

Code in `conda.base` is the lowest level of the application stack. It is loaded and executed virtually every time the application is executed. Any code within, and any of its imports, must be highly performant.

Conda modules importable from `conda.base` are

- `conda.base`
- `conda.common`

Modules prohibited from importing `conda.base` are:

- `conda.common`

All other `conda` modules may import from `conda.base`.

### constants

This file should hold most string literals and magic numbers used throughout the code base. The exception is if a literal is specifically meant to be private to and isolated within a module. Think of this as a "more static" source of configuration information.

Another important source of "static" configuration is conda/models/enums.py.

#### Classes

| | |
|---|---|
| *SafetyChecks* | Create a collection of name/value pairs. |
| *PathConflict* | Create a collection of name/value pairs. |
| *DepsModifier* | Flags to enable alternate handling of dependencies. |
| *UpdateModifier* | Create a collection of name/value pairs. |
| *ChannelPriorityMeta* | Metaclass for Enum |
| *ValueEnum* | Subclass of enum that returns the value of the enum as its str representation |
| *ChannelPriority* | Subclass of enum that returns the value of the enum as its str representation |
| *SatSolverChoice* | Subclass of enum that returns the value of the enum as its str representation |
| *NoticeLevel* | Subclass of enum that returns the value of the enum as its str representation |

### Attributes

| |
|---|
| *PREFIX_PLACEHOLDER* |
| *machine_bits* |
| *APP_NAME* |
| *SEARCH_PATH* |
| *SEARCH_PATH* |
| *DEFAULT_CHANNEL_ALIAS* |
| *CONDA_HOMEPAGE_URL* |
| *ERROR_UPLOAD_URL* |
| *DEFAULTS_CHANNEL_NAME* |
| *PLATFORMS* |
| *KNOWN_SUBDIRS* |
| *PLATFORM_DIRECTORIES* |
| *RECOGNIZED_URL_SCHEMES* |
| *DEFAULT_CHANNELS_UNIX* |
| *DEFAULT_CHANNELS_WIN* |
| *DEFAULT_CUSTOM_CHANNELS* |
| *DEFAULT_CHANNELS* |
| *ROOT_ENV_NAME* |
| *UNUSED_ENV_NAME* |
| *ROOT_NO_RM* |
| *DEFAULT_AGGRESSIVE_UPDATE_PACKAGES* |
| *COMPATIBLE_SHELLS* |
| *COMPATIBLE_SHELLS* |
| *MAX_CHANNEL_PRIORITY* |
| *CONDA_PACKAGE_EXTENSION_V1* |

continues on next page

Table 1 – continued from previous page

| |
|---|
| *CONDA_PACKAGE_EXTENSION_V2* |
| *CONDA_PACKAGE_EXTENSIONS* |
| *CONDA_PACKAGE_PARTS* |
| *CONDA_TARBALL_EXTENSION* |
| *CONDA_TEMP_EXTENSION* |
| *CONDA_TEMP_EXTENSIONS* |
| *CONDA_LOGS_DIR* |
| *UNKNOWN_CHANNEL* |
| *REPODATA_FN* |
| *NOTICES_FN* |
| *NOTICES_CACHE_FN* |
| *NOTICES_CACHE_SUBDIR* |
| *NOTICES_DECORATOR_DISPLAY_INTERVAL* |
| *DRY_RUN_PREFIX* |
| *PREFIX_NAME_DISALLOWED_CHARS* |
| *DEFAULT_SOLVER* |
| *CLASSIC_SOLVER* |
| *DEFAULT_JSON_REPORTER_BACKEND* |
| *DEFAULT_CONSOLE_REPORTER_BACKEND* |
| *DEFAULT_CONDA_LIST_FIELDS* |
| *CONDA_LIST_FIELDS* |
| *PACKAGE_CACHE_MAGIC_FILE* |
| *PREFIX_MAGIC_FILE* |
| *PREFIX_FROZEN_FILE* |
| *PREFIX_STATE_FILE* |
| *PACKAGE_ENV_VARS_DIR* |

continues on next page

Table 1 – continued from previous page

| |
|---|
| *CONDA_ENV_VARS_UNSET_VAR* |
| *NAMESPACES_MAP* |
| *NAMESPACE_PACKAGE_NAMES* |
| *NAMESPACES* |
| *NO_PLUGINS* |

PREFIX_PLACEHOLDER: Final = '/opt/anaconda1anaconda2anaconda3'

machine_bits:  Final

APP_NAME: Final = 'conda'

SEARCH_PATH: tuple[str, Ellipsis]

SEARCH_PATH = ('C:/ProgramData/conda/.condarc', 'C:/ProgramData/conda/condarc',
'C:/ProgramData/conda/condarc.d')

DEFAULT_CHANNEL_ALIAS: Final = 'https://conda.anaconda.org'

CONDA_HOMEPAGE_URL: Final = 'https://conda.io'

ERROR_UPLOAD_URL: Final = 'https://conda.io/conda-post/unexpected-error'

DEFAULTS_CHANNEL_NAME: Final = 'defaults'

PLATFORMS: Final = ('emscripten-wasm32', 'wasi-wasm32', 'freebsd-64', 'linux-32',
'linux-64', 'linux-aarch64',...

KNOWN_SUBDIRS: Final = ('noarch',)

PLATFORM_DIRECTORIES

RECOGNIZED_URL_SCHEMES: Final = ('http', 'https', 'ftp', 's3', 'file')

DEFAULT_CHANNELS_UNIX: Final = ('https://repo.anaconda.com/pkgs/main',
'https://repo.anaconda.com/pkgs/r')

DEFAULT_CHANNELS_WIN: Final = ('https://repo.anaconda.com/pkgs/main',
'https://repo.anaconda.com/pkgs/r',...

DEFAULT_CUSTOM_CHANNELS: Final

DEFAULT_CHANNELS: Final

ROOT_ENV_NAME: Final = 'base'

UNUSED_ENV_NAME: Final = 'unused-env-name'

ROOT_NO_RM: Final = ('python', 'pycosat', 'ruamel.yaml', 'conda', 'openssl', 'requests')

DEFAULT_AGGRESSIVE_UPDATE_PACKAGES: Final = ('ca-certificates', 'certifi', 'openssl')

COMPATIBLE_SHELLS: tuple[str, Ellipsis]

COMPATIBLE_SHELLS = ('bash', 'cmd.exe', 'fish', 'tcsh', 'xonsh', 'zsh', 'powershell')

MAX_CHANNEL_PRIORITY: Final = 10000

CONDA_PACKAGE_EXTENSION_V1: Final = '.tar.bz2'

CONDA_PACKAGE_EXTENSION_V2: Final = '.conda'

CONDA_PACKAGE_EXTENSIONS: Final = ()

CONDA_PACKAGE_PARTS: Final

CONDA_TARBALL_EXTENSION: Final

CONDA_TEMP_EXTENSION: Final = '.c~'

CONDA_TEMP_EXTENSIONS: Final = ()

CONDA_LOGS_DIR: Final = '.logs'

UNKNOWN_CHANNEL: Final = '<unknown>'

REPODATA_FN: Final = 'repodata.json'

NOTICES_FN: Final = 'notices.json'

NOTICES_CACHE_FN: Final = 'notices.cache'

NOTICES_CACHE_SUBDIR: Final = 'notices'

NOTICES_DECORATOR_DISPLAY_INTERVAL: Final = 86400

DRY_RUN_PREFIX: Final = 'Dry run action:'

PREFIX_NAME_DISALLOWED_CHARS: Final

class SafetyChecks(*args, **kwds)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**disabled = 'disabled'**

**warn = 'warn'**

**enabled = 'enabled'**

**__str__**() → str

Return str(self).

**class PathConflict**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**clobber = 'clobber'**

**warn = 'warn'**

**prevent = 'prevent'**

**__str__**() → str

Return str(self).

**class DepsModifier**(**args*, ***kwds*)

Bases: enum.Enum

Flags to enable alternate handling of dependencies.

**NOT_SET = 'not_set'**

**NO_DEPS = 'no_deps'**

**ONLY_DEPS = 'only_deps'**

**__str__**() → str

Return str(self).

**class UpdateModifier**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

  ```
  >>> Color.RED
  <Color.RED: 1>
  ```

- value lookup:

  ```
  >>> Color(1)
  <Color.RED: 1>
  ```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**SPECS_SATISFIED_SKIP_SOLVE = 'specs_satisfied_skip_solve'**

**FREEZE_INSTALLED = 'freeze_installed'**

**UPDATE_DEPS = 'update_deps'**

**UPDATE_SPECS = 'update_specs'**

**UPDATE_ALL = 'update_all'**

**__str__**() → str

Return str(self).

**class ChannelPriorityMeta**

Bases: enum.EnumMeta

Metaclass for Enum

**__call__**(*value*, **args*, ***kwargs*)

Either returns an existing member, or creates a new enum class.

This method is used both when an enum class is given a value to match to an enumeration member (i.e. Color(3)) and for the functional API (i.e. Color = Enum('Color', names='RED GREEN BLUE')).

The value lookup branch is chosen if the enum is final.

When used for the functional API:

*value* will be the name of the new class.

*names* should be either a string of white-space/comma delimited names (values will start at *start*), or an iterator/mapping of name, value pairs.

*module* should be set to the module this class is being created in; if it is not set, an attempt to find that module will be made, but if it fails the class will not be picklable.

*qualname* should be set to the actual location this class can be found at in its module; by default it is set to the global scope. If this is not correct, unpickling will fail in some circumstances.

*type*, if set, will be mixed in as the first base class.

**class ValueEnum**(**args*, ***kwds*)

Bases: enum.Enum

Subclass of enum that returns the value of the enum as its str representation

__str__() → str
Return str(self).

class ChannelPriority(*args, **kwds)
Bases: ValueEnum

Subclass of enum that returns the value of the enum as its str representation

__name__ = 'ChannelPriority'

STRICT = 'strict'

FLEXIBLE = 'flexible'

DISABLED = 'disabled'

class SatSolverChoice(*args, **kwds)
Bases: ValueEnum

Subclass of enum that returns the value of the enum as its str representation

PYCOSAT = 'pycosat'

PYCRYPTOSAT = 'pycryptosat'

PYSAT = 'pysat'

DEFAULT_SOLVER: Final = 'libmamba'

CLASSIC_SOLVER: Final = 'classic'

DEFAULT_JSON_REPORTER_BACKEND: Final = 'json'

DEFAULT_CONSOLE_REPORTER_BACKEND: Final = 'classic'

DEFAULT_CONDA_LIST_FIELDS: Final = ('name', 'version', 'build', 'channel_name')

CONDA_LIST_FIELDS: Final

class NoticeLevel(*args, **kwds)
Bases: ValueEnum

Subclass of enum that returns the value of the enum as its str representation

CRITICAL = 'critical'

WARNING = 'warning'

INFO = 'info'

PACKAGE_CACHE_MAGIC_FILE: Final[conda.common.path.PathType] = 'urls.txt'

PREFIX_MAGIC_FILE: Final[conda.common.path.PathType]

PREFIX_FROZEN_FILE: Final[conda.common.path.PathType]

PREFIX_STATE_FILE: Final[conda.common.path.PathType]

PACKAGE_ENV_VARS_DIR: Final[conda.common.path.PathType]

CONDA_ENV_VARS_UNSET_VAR: Final = '***unset***'

**NAMESPACES_MAP: Final**

**NAMESPACE_PACKAGE_NAMES: Final**

**NAMESPACES: Final**

**NO_PLUGINS: Final = False**

### context

Conda's global configuration object.

The context aggregates all configuration files, environment variables, and command line arguments into one global stateful object to be used across all of conda.

## Classes

| |
|---|
| *Context* |
| *ContextStackObject* |
| *ContextStack* |

### Functions

| | |
|---|---|
| *user_data_dir*(→ conda.common.path.PathType) | |
| *mockable_context_envs_dirs*(...) | |
| *channel_alias_validation*(→ str \| Literal[True]) | |
| *default_python_default*(→ str) | |
| *default_python_validation*(→ str \| Literal[True]) | |
| *list_fields_validation*(→ str \| Literal[True]) | |
| *ssl_verify_validation*(→ str \| Literal[True]) | |
| *_warn_defaults_deprecation*(→ None) | |
| *reset_context*(→ Context) | |
| *fresh_context*(→ collections.abc.Iterator[Context]) | |
| *stack_context*(→ None) | |
| *stack_context_default*(→ None) | |
| *replace_context*(→ None) | |
| *replace_context_default*(→ None) | |
| *env_name*(→ conda.common.path.PathType \| str \| None) | |
| *locate_prefix_by_name*(→ conda.common.path.PathType) | Find the location of a prefix given a conda env name. If the location does not exist, an |
| *validate_channels*(→ tuple[str, Ellipsis]) | Validate if the given channel URLs are allowed based on the context's allowlist |
| *validate_prefix_name*(→ conda.common.path.PathType) | Run various validations to make sure prefix_name is valid |
| *determine_target_prefix*(→ conda.common.path.PathType) | Get the prefix to operate in. The prefix may not yet exist. |
| *_first_writable_envs_dir*(→ conda.common.path.PathType) | |
| *get_plugin_config_data*(→ dict[pathlib.Path, dict[str, ...]) | |
| *add_plugin_setting*() → None) | |
| *remove_all_plugin_settings*(→ None) | |

### Attributes

| |
|---|
| *_platform_map* |
| *non_x86_machines* |
| *_arch_names* |
| *user_rc_path* |
| *sys_rc_path* |
| *context_stack* |
| *conda_tests_ctxt_mgmt_def_pol* |
| *context* |

**_platform_map**

**non_x86_machines**

**_arch_names**

**user_rc_path: conda.common.path.PathType**

**sys_rc_path: conda.common.path.PathType**

**user_data_dir**(*appname: str | None = None*, *appauthor: str | None | Literal[False] = None*, *version: str | None = None*, *roaming: bool = False*) → conda.common.path.PathType

**mockable_context_envs_dirs**(*root_writable: bool*, *root_prefix: conda.common.path.PathType*, *_envs_dirs: conda.common.path.PathsType*) → tuple[conda.common.path.PathType, Ellipsis]

**channel_alias_validation**(*value: str*) → str | Literal[True]

**default_python_default**() → str

**default_python_validation**(*value: str*) → str | Literal[True]

**list_fields_validation**(*value: collections.abc.Iterable[str]*) → str | Literal[True]

**ssl_verify_validation**(*value: str*) → str | Literal[True]

**_warn_defaults_deprecation**() → None

**class Context**(*search_path: conda.common.path.PathsType | None = None*, *argparse_args: argparse.Namespace | None = None*, ***kwargs*)

Bases: *conda.common.configuration.Configuration*

**property pip_interop_enabled**

**property plugin_manager: *conda.plugins.manager.CondaPluginManager***

This is the preferred way of accessing the `PluginManager` object for this application and is located here to avoid problems with cyclical imports elsewhere in the code.

property conda_build_local_paths: tuple[conda.common.path.PathType, Ellipsis]

property conda_build_local_urls: tuple[str, Ellipsis]

property croot: conda.common.path.PathType

This is where source caches and work folders live

property local_build_root: conda.common.path.PathType

property conda_build: dict[str, Any]

property arch_name: str

property platform: str

property default_threads: int | None

property repodata_threads: int | None

property fetch_threads: int | None

If both are not overriden (0), return experimentally-determined value of 5

property verify_threads: int | None

property execute_threads: int | None

property subdir: str

property subdirs: tuple[str, str]

property bits: int

property root_writable: bool

property envs_dirs: tuple[conda.common.path.PathType, Ellipsis]

property pkgs_dirs: tuple[conda.common.path.PathType, Ellipsis]

property default_prefix: conda.common.path.PathType

property active_prefix: conda.common.path.PathType

property shlvl: int

property aggressive_update_packages: tuple[*conda.models.match_spec.MatchSpec*, Ellipsis]

property target_prefix: conda.common.path.PathType

property conda_prefix: conda.common.path.PathType

property conda_exe: conda.common.path.PathType

property av_data_dir: conda.common.path.PathType

Where critical artifact verification data (e.g., various public keys) can be found.

property signing_metadata_url_base: str | None

Base URL for artifact verification signing metadata (*.root.json, key_mgr.json).

property conda_exe_vars_dict: dict[str, str | None]

The vars can refer to each other if necessary since the dict is ordered. None means unset it.

property migrated_channel_aliases: tuple[*conda.models.channel.Channel*, Ellipsis]

property prefix_specified: bool

property restore_free_channel: bool

property channels: tuple[str, Ellipsis]

property config_files: tuple[conda.common.path.PathType, Ellipsis]

property use_only_tar_bz2: bool

property binstar_upload: bool | None

property trace: bool

Alias for context.verbosity >=4.

property debug: bool

Alias for context.verbosity >=3.

property info: bool

Alias for context.verbosity >=2.

property verbose: bool

Alias for context.verbosity >=1.

property verbosity: int

Verbosity level.

**For cleaner and readable code it is preferable to use the following alias properties:**
context.trace context.debug context.info context.verbose context.log_level

property log_level: int

Map context.verbosity to logging level.

property console: str

property auto_activate_base: bool

property default_activation_env: str

property category_map: dict[str, tuple[str, Ellipsis]]

add_pip_as_python_dependency

allow_conda_downgrades

allow_cycles

allow_softlinks

auto_update_conda

auto_activate

_default_activation_env

auto_stack

notify_outdated_conda

clobber

changeps1

env_prompt

environment_specifier

create_default_packages

register_envs

protect_frozen_envs

default_python

download_only

enable_private_envs

force_32bit

non_admin_enabled

prefix_data_interoperability

_default_threads

_repodata_threads

_fetch_threads

_verify_threads

_execute_threads

_aggressive_update_packages

safety_checks

extra_safety_checks

_signing_metadata_url_base

path_conflict

pinned_packages

disallowed_packages

rollback_enabled

track_features

use_index_cache

separate_format_cache

_root_prefix

_envs_dirs

_pkgs_dirs

_subdir

_subdirs

local_repodata_ttl

ssl_verify

client_ssl_cert

client_ssl_cert_key

proxy_servers

remote_connect_timeout_secs

remote_read_timeout_secs

remote_max_retries

remote_backoff_factor

add_anaconda_token

allow_non_channel_urls

_channel_alias

channel_priority

_channels

channel_settings

_custom_channels

_custom_multichannels

_default_channels

_migrated_channel_aliases

migrated_custom_channels

override_channels_enabled

show_channel_urls

use_local

allowlist_channels

denylist_channels

_restore_free_channel

repodata_fns

_use_only_tar_bz2

always_softlink

always_copy

always_yes

_debug

_trace

dev

dry_run

error_upload_url

force

json

_console

list_fields

offline

quiet

ignore_pinned

report_errors

shortcuts

number_channel_notices

shortcuts

shortcuts_only

_verbosity

experimental

no_lock

repodata_use_zst

envvars_force_uppercase

deps_modifier

update_modifier

sat_solver

solver_ignore_timestamps

**solver**

**force_remove**

**force_reinstall**

**target_prefix_override**

**unsatisfiable_hints**

**unsatisfiable_hints_check_depth**

**bld_path**

**anaconda_upload**

**_croot**

**_conda_build**

**no_plugins**

**post_build_validation**() → list[*conda.common.configuration.ValidationError*]

**plugins**() → *conda.plugins.config.PluginConfig*

Preferred way of accessing settings introduced by the settings plugin hook

**_native_subdir**() → str

**known_subdirs**() → set[str]

**trash_dir**() → conda.common.path.PathType

**root_prefix**() → conda.common.path.PathType

**channel_alias**() → *conda.models.channel.Channel*

**default_channels**() → list[*conda.models.channel.Channel*]

**custom_multichannels**() → dict[str, tuple[*conda.models.channel.Channel*, Ellipsis]]

**custom_channels**() → dict[str, *conda.models.channel.Channel*]

**solver_user_agent**() → str

**user_agent**() → str

**_override**(*key: str*, *value: Any*) → collections.abc.Iterator[None]

TODO: This might be broken in some ways. Unsure what happens if the *old* value is a property and gets set to a new value. Or if the new value overrides the validation logic on the underlying ParameterLoader instance.

Investigate and implement in a safer way.

**requests_version**() → str

**python_implementation_name_version**() → tuple[str, str]

**platform_system_release**() → tuple[str, str]

**os_distribution_name_version**() → tuple[str, str]

**libc_family_version**() → tuple[str | None, str | None]

**get_descriptions**() → dict[str, str]

**description_map**() → dict[str, str]

**reset_context**(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → *Context*

**fresh_context**(*env: dict[str, str] | None = None*, *search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*, ***kwargs*) → collections.abc.Iterator[*Context*]

**class ContextStackObject**(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*)

**set_value**(*search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

**apply**()

**class ContextStack**

**push**(*search_path: conda.common.path.PathsType*, *argparse_args: argparse.Namespace | None*) → None

**apply**()

**pop**()

**replace**(*search_path: conda.common.path.PathsType*, *argparse_args: argparse.Namespace | None*) → None

**context_stack**

**stack_context**(*pushing: bool*, *search_path: conda.common.path.PathsType = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

**stack_context_default**(*pushing: bool*, *argparse_args: argparse.Namespace | None = None*) → None

**replace_context**(*pushing: bool | None = None*, *search_path: collections.abc.Iterable[str] = SEARCH_PATH*, *argparse_args: argparse.Namespace | None = None*) → None

**replace_context_default**(*pushing: bool | None = None*, *argparse_args: argparse.Namespace | None = None*) → None

**conda_tests_ctxt_mgmt_def_pol**

**env_name**(*prefix: conda.common.path.PathType*) → conda.common.path.PathType | str | None

**locate_prefix_by_name**(*name: str*, *envs_dirs: conda.common.path.PathsType | None = None*) → conda.common.path.PathType

Find the location of a prefix given a conda env name. If the location does not exist, an error is raised.

**validate_channels**(*channels: collections.abc.Iterator[str]*) → tuple[str, Ellipsis]

Validate if the given channel URLs are allowed based on the context's allowlist and denylist configurations.

**Parameters**

**channels** -- A list of channels (either URLs or names) to validate.

**Raises**

- *ChannelNotAllowed* -- If any URL is not in the allowlist.
- *ChannelDenied* -- If any URL is in the denylist.

**validate_prefix_name**(*prefix_name: str*, *ctx:* Context, *allow_base: bool = True*) → conda.common.path.PathType

Run various validations to make sure prefix_name is valid

**determine_target_prefix**(*ctx:* Context, *args: argparse.Namespace | None = None*) → conda.common.path.PathType

Get the prefix to operate in. The prefix may not yet exist.

**Parameters**

- **ctx** -- the context of conda
- **args** -- the argparse args from the command line

Returns: the prefix Raises: CondaEnvironmentNotFoundError if the prefix is invalid

**_first_writable_envs_dir**() → conda.common.path.PathType

**get_plugin_config_data**(*data: dict[pathlib.Path, dict[str,* conda.common.configuration.RawParameter*]]*) → dict[pathlib.Path, dict[str, *conda.common.configuration.RawParameter*]]

**add_plugin_setting**(*name: str*, *parameter:* conda.common.configuration.Parameter, *aliases: tuple[str, Ellipsis] = ()*) → None

**remove_all_plugin_settings**() → None

**context**

### cli

### actions

Collection of custom argparse actions.

## Classes

| | |
|---|---|
| *NullCountAction* | Information about how to convert command line strings to Python objects. |
| *ExtendConstAction* | A derivative of _AppendConstAction and Python 3.8's _ExtendAction |

**class NullCountAction**(*option_strings*, *dest*, *default=None*, *required=False*, *help=None*)

Bases: argparse._CountAction

Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

**Keyword Arguments**

- **which** (*- option_strings -- A list of command-line option strings*) -- should be associated with this action.
- **object** (*- dest -- The name of the attribute to hold the created*)
- **be** (*- nargs -- The number of command-line arguments that should*) -- consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **the** (*- metavar -- The name to be used for the option's argument with*) -- option uses an action that takes no values.
- **specified.** (*- default -- The value to be produced if the option is not*)
- **and** (*- type -- A callable that accepts a single string argument,*) -- returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **None,** (*- choices -- A container of values that should be allowed. If not*) -- after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **the** -- command line. This is only meaningful for optional command-line arguments.
- **argument.** (*- help -- The help string describing the*)
- **the** -- help string. If None, the 'dest' value will be used as the name.

**static _ensure_value**(*namespace*, *name*, *value*)

**__call__**(*parser*, *namespace*, *values*, *option_string=None*)

**class ExtendConstAction**(*option_strings*, *dest*, *const*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

A derivative of _AppendConstAction and Python 3.8's _ExtendAction

**__call__**(*parser*, *namespace*, *values*, *option_string=None*)

### common

Common utilities for conda command line tools.

## Functions

| | |
|---|---|
| confirm([message, choices, default, dry_run]) | |
| confirm_yn([message, default, dry_run]) | |
| is_active_prefix(→ bool) | Determines whether the args we pass in are pointing to the active prefix. |
| arg2spec(arg[, json, update]) | |
| specs_from_args(args[, json]) | |
| strip_comment(line) | |
| spec_from_line(line) | |
| specs_from_url(url[, json]) | |
| names_in_specs(names, specs) | |
| disp_features(features) | |
| stdout_json(d) | |
| stdout_json_success([success]) | |
| print_envs_list(known_conda_prefixes[, output]) | |
| check_non_admin() | |
| validate_prefix(→ str) | Verifies the prefix is a valid conda environment. |
| validate_prefix_is_writable(→ str) | Verifies the environment directory is writable by trying to access |
| validate_subdir_config() | Validates that the configured subdir is ok. A subdir that is different from |
| print_activate(env_name_or_prefix) | |
| validate_environment_files_consistency(→ None) | Validates that all the provided environment files are of the same format type. |
| validate_file_exists(filename) | Validate the existence of an environment file. |

### Attributes

*spec_pat*

**confirm**(*message='Proceed'*, *choices=('yes', 'no')*, *default='yes'*, *dry_run=NULL*)

**confirm_yn**(*message='Proceed'*, *default='yes'*, *dry_run=NULL*)

**is_active_prefix**(*prefix: str*) → bool

Determines whether the args we pass in are pointing to the active prefix. Can be used a validation step to make sure operations are not being performed on the active prefix.

**arg2spec**(*arg*, *json=False*, *update=False*)

**specs_from_args**(*args*, *json=False*)

**spec_pat**

**strip_comment**(*line*)

**spec_from_line**(*line*)

**specs_from_url**(*url*, *json=False*)

**names_in_specs**(*names*, *specs*)

**disp_features**(*features*)

**stdout_json**(*d*)

**stdout_json_success**(*success=True*, ***kwargs*)

**print_envs_list**(*known_conda_prefixes*, *output=True*)

**check_non_admin**()

**validate_prefix**(*prefix*) → str

Verifies the prefix is a valid conda environment.

**Raises**

- ***EnvironmentLocationNotFound*** -- Non-existent path or not a directory.
- ***DirectoryNotACondaEnvironmentError*** -- Directory is not a conda environment.

**Returns**

Valid prefix.

**Return type**

str

**validate_prefix_is_writable**(*prefix: str*) → str

Verifies the environment directory is writable by trying to access the conda-meta/history file. If this file is not writable then we assume the whole prefix is not writable and raise an exception.

**Raises**

***EnvironmentNotWritableError*** -- Conda does not have permission to write to the prefix

**Returns**
Valid prefix.

**Return type**
str

`validate_subdir_config()`

Validates that the configured subdir is ok. A subdir that is different from the native system is only allowed if it comes from the global configuration, or from an environment variable.

**Raises**
*OperationNotAllowed* -- Active environment is not allowed to request non-native platform packages

`print_activate(env_name_or_prefix)`

`validate_environment_files_consistency(files: list[str]) → None`

Validates that all the provided environment files are of the same format type.

This function checks if all provided environment files are of the same format type using the conda plugin system's environment specifiers. It prevents mixing different environment file formats (e.g., YAML, explicit package lists, requirements.txt).

**Raises**
*EnvironmentFileTypeMismatchError* -- When files with different formats are found

`validate_file_exists(filename: str)`

Validate the existence of an environment file.

This function checks if the given `filename` exists as an environment file. If the *filename* has a URL scheme supported by `CONDA_SESSION_SCHEMES`, it assumes the file is accessible and returns without further validation. Otherwise, it expands the given path and verifies its existence. If the file does not exist, an `EnvironmentFileNotFound` exception is raised.

**Parameters**
**filename** (*str*) -- The path or URL of the environment file to validate.

**Raises**
*EnvironmentFileNotFound* -- If the file does not exist and is not a valid URL.

## conda_argparse

Conda command line interface parsers.

### Classes

| | |
|---|---|
| *ArgumentParser* | Object for parsing command line strings into Python objects. |
| *_GreedySubParsersAction* | A custom subparser action to conditionally act as a greedy consumer. |

### Functions

| | |
|---|---|
| *generate_pre_parser*(→ ArgumentParser) | |
| *generate_parser*(→ ArgumentParser) | |
| *do_call*(args, parser) | Serves as the primary entry point for commands referred to in this file and for |
| *find_builtin_commands*(parser) | |
| *_exec*(executable_args, env_vars) | |
| *_exec_win*(executable_args, env_vars) | |
| *_exec_unix*(executable_args, env_vars) | |
| *configure_parser_plugins*(→ None) | For each of the provided plugin-based subcommands, we'll create |

### Attributes

| |
|---|
| *escaped_user_rc_path* |
| *escaped_sys_rc_path* |
| *BUILTIN_COMMANDS* |

**escaped_user_rc_path**

**escaped_sys_rc_path**

**BUILTIN_COMMANDS**

**generate_pre_parser**(***kwargs*) → *ArgumentParser*

**generate_parser**(***kwargs*) → *ArgumentParser*

**do_call**(*args: argparse.Namespace*, *parser:* ArgumentParser)

Serves as the primary entry point for commands referred to in this file and for all registered plugin subcommands.

**find_builtin_commands**(*parser*)

**class ArgumentParser**(**args*, *add_help=True*, ***kwargs*)

Bases: `argparse.ArgumentParser`

Object for parsing command line strings into Python objects.

**Keyword Arguments**

- **(default** (`- usage -- A usage message`) -- `os.path.basename(sys.argv[0])`)
- **(default** -- auto-generated from arguments)

- **does** (- *description -- A description of what the program*)
- **descriptions** (- *epilog -- Text following the argument*)
- **one** (- *parents -- Parsers whose arguments should be copied into this*)
- **messages** (- *formatter_class -- HelpFormatter class for printing help*)
- **arguments** (- *argument_default -- The default value for all*)
- **containing** (- *fromfile_prefix_chars -- Characters that prefix files*) -- additional arguments
- **arguments**
- **conflicts** (- *conflict_handler -- String indicating how to handle*)
- **option** (- *add_help -- Add a -h/-help*)
- **unambiguously** (- *allow_abbrev -- Allow long options to be abbreviated*)
- **with** (- *exit_on_error -- Determines whether or not ArgumentParser exits*) -- error info when an error occurs

**_check_value**(*action*, *value*)

**parse_args**(**args*, *override_args=None*, ***kwargs*)

**class _GreedySubParsersAction**(*option_strings*, *prog*, *parser_class*, *dest=SUPPRESS*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse._SubParsersAction`

A custom subparser action to conditionally act as a greedy consumer.

This is a workaround since argparse.REMAINDER does not work as expected, see https://github.com/python/cpython/issues/61252.

**__call__**(*parser*, *namespace*, *values*, *option_string=None*)

**_get_subactions**()

Sort actions for subcommands to appear alphabetically in help blurb.

**_exec**(*executable_args*, *env_vars*)

**_exec_win**(*executable_args*, *env_vars*)

**_exec_unix**(*executable_args*, *env_vars*)

**configure_parser_plugins**(*sub_parsers*) → None

For each of the provided plugin-based subcommands, we'll create a new subparser for an improved help printout and calling the *configure_parser()* with the newly created subcommand specific argument parser.

## find_commands

Utilities for finding executables and *conda-** commands.

### Functions

| | |
|---|---|
| *find_executable*(executable[, include_others]) | |
| *find_commands*([include_others]) | |

**find_executable**(*executable*, *include_others=True*)

**find_commands**(*include_others=True*)

## helpers

Collection of helper functions to standardize reused CLI arguments.

### Classes

| | |
|---|---|
| *LazyChoicesAction* | Information about how to convert command line strings to Python objects. |
| *_ValidatePackages* | Used to validate match specs of packages |

### Functions

| | |
|---|---|
| add_parser_create_install_update(p[, pre-fix_required]) | |
| add_parser_pscheck(→ None) | |
| add_parser_show_channel_urls(→ None) | |
| add_parser_help(→ None) | So we can use consistent capitalization and periods in the help. You must |
| add_parser_prefix(→ parse._MutuallyExclusiveGroup) | arg- |
| add_parser_prefix_to_group(→ None) | |
| add_parser_json(→ argparse._ArgumentGroup) | |
| add_output_and_prompt_options(→ parse._ArgumentGroup) | arg- |
| add_parser_frozen_env(p) | |
| add_parser_channels(→ parse._ArgumentGroup) | arg- |
| add_parser_solver_mode(→ parse._ArgumentGroup) | arg- |
| add_parser_update_modifiers(solver_mode_option | |
| add_parser_prune(→ None) | |
| add_parser_solver(→ None) | Add a command-line flag for alternative solver backends. |
| add_parser_networking(→ parse._ArgumentGroup) | arg- |
| add_parser_package_install_options(...) | |
| add_parser_known(→ None) | |
| add_parser_default_packages(→ None) | |
| add_parser_platform(parser) | |
| add_parser_verbose(→ None) | |
| add_parser_environment_specifier(→ None) | |
| comma_separated_stripped(→ list[str]) | Custom type for argparse to handle comma-separated strings with stripping |

**class LazyChoicesAction**(*option_strings*, *dest*, *choices_func*, ***kwargs*)

Bases: argparse.Action

Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all

attributes of Action instances.

**Keyword Arguments**

- **which** (- *option_strings -- A list of command-line option strings*) -- should be associated with this action.
- **object** (- *dest -- The name of the attribute to hold the created*)
- **be** (- *nargs -- The number of command-line arguments that should*) -- consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **the** (- *metavar -- The name to be used for the option's argument with*) -- option uses an action that takes no values.
- **specified.** (- *default -- The value to be produced if the option is not*)
- **and** (- *type -- A callable that accepts a single string argument,*) -- returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **None,** (- *choices -- A container of values that should be allowed. If not*) -- after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **the** -- command line. This is only meaningful for optional command-line arguments.
- **argument.** (- *help -- The help string describing the*)
- **the** -- help string. If None, the 'dest' value will be used as the name.

**__call__**(*parser*, *namespace*, *values*, *option_string=None*)

**class _ValidatePackages**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse._StoreAction

Used to validate match specs of packages

**static _validate_no_denylist_channels**(*packages_specs*)

Ensure the packages do not contain denylist_channels

**__call__**(*parser*, *namespace*, *values*, *option_string=None*)

**add_parser_create_install_update**(*p*, *prefix_required=False*)

**add_parser_pscheck**(*p: argparse.ArgumentParser*) → None

**add_parser_show_channel_urls**(*p: argparse.ArgumentParser | argparse._ArgumentGroup*) → None

**add_parser_help**(*p: argparse.ArgumentParser*) → None

So we can use consistent capitalization and periods in the help. You must use the add_help=False argument to ArgumentParser or add_parser to use this. Add this first to be consistent with the default argparse output.

**add_parser_prefix**(*p: argparse.ArgumentParser*, *prefix_required: bool = False*) → argparse._MutuallyExclusiveGroup

**add_parser_prefix_to_group**(*m: argparse._MutuallyExclusiveGroup*) → None

**add_parser_json**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_output_and_prompt_options**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_parser_frozen_env**(*p: argparse.ArgumentParser*)

**add_parser_channels**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_parser_solver_mode**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_parser_update_modifiers**(*solver_mode_options: argparse.ArgumentParser*)

**add_parser_prune**(*p: argparse.ArgumentParser*) → None

**add_parser_solver**(*p: argparse.ArgumentParser*) → None

Add a command-line flag for alternative solver backends.

See `context.solver` for more info.

**add_parser_networking**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_parser_package_install_options**(*p: argparse.ArgumentParser*) → argparse._ArgumentGroup

**add_parser_known**(*p: argparse.ArgumentParser*) → None

**add_parser_default_packages**(*p: argparse.ArgumentParser*) → None

**add_parser_platform**(*parser*)

**add_parser_verbose**(*parser: argparse.ArgumentParser | argparse._ArgumentGroup*) → None

**add_parser_environment_specifier**(*p: argparse.ArgumentParser*) → None

**comma_separated_stripped**(*value: str*) → list[str]

Custom type for argparse to handle comma-separated strings with stripping

### install

Conda package installation logic.

Core logic for *conda [create|install|update|remove]* commands.

See conda.cli.main_create, conda.cli.main_install, conda.cli.main_update, and conda.cli.main_remove for the entry points into this module.

### Classes

| | |
|---|---|
| *TryRepodata* | |
| *Repodatas* | |

### Functions

| | |
|---|---|
| *validate_prefix_exists*(→ None) | Validate that we are receiving at least one valid value for --name or --prefix. |
| *validate_new_prefix*(→ str) | Ensure that the new prefix does not exist. |
| *check_prefix*(prefix[, json]) | |
| *clone*(src_arg, dst_prefix[, json, quiet, index_args]) | |
| *print_activate*(env_name_or_prefix) | |
| *get_revision*(arg[, json]) | |
| *get_index_args*(→ dict[str, any]) | Returns a dict of args required for fetching an index |
| *validate_install_command*(prefix[, command]) | Executes a set of validations that are required before any installation |
| *ensure_update_specs_exist*(prefix, specs) | Checks that each spec that is requested as an update exists in the prefix |
| *install_clone*(args, parser) | Executes an install of a new conda environment by cloning. |
| *install*(args, parser[, command]) | Logic for *conda install*, *conda update*, *conda remove*, and *conda create*. |
| *install_revision*(args, parser) | Install a previous version of a conda environment |
| *revert_actions*(prefix[, revision, index]) | |
| *handle_txn*(unlink_link_transaction, prefix, args, newenv) | |

### Attributes

| | |
|---|---|
| *stderrlog* | |

#### stderrlog

**validate_prefix_exists**(*prefix: str | pathlib.Path*) → None

Validate that we are receiving at least one valid value for --name or --prefix.

**validate_new_prefix**(*dest: str*, *force: bool = False*) → str

Ensure that the new prefix does not exist.

**check_prefix**(*prefix: str*, *json=False*)

**clone**(*src_arg*, *dst_prefix*, *json=False*, *quiet=False*, *index_args=None*)

**print_activate**(*env_name_or_prefix*)

**get_revision**(*arg*, *json=False*)

**get_index_args**(*args*) → dict[str, any]

Returns a dict of args required for fetching an index :param args: The args provided by the cli :returns: dict of index args

**class TryRepodata**(*notify_success*, *repodata*, *last_repodata*, *index_args*, *allowed_errors*)

**__enter__**()

**__exit__**(*exc_type*, *exc_value*, *traceback*)

**class Repodatas**(*repodata_fns*, *index_args*, *allows_errors=()*)

**__iter__**()

**succeed**()

**validate_install_command**(*prefix: str*, *command: str = 'install'*)

Executes a set of validations that are required before any installation command is executed. This includes:

- ensure the configuration is valid
- ensuring the user in not an admin
- ensure the user is not forcing 32bit installs in the root prefix

**Parameters**

- **prefix** -- The prefix where the environment will be created
- **command** -- Type of operation being performed

**Raises**

error if the configuration for the install is bad

**ensure_update_specs_exist**(*prefix: str*, *specs: list[str]*)

Checks that each spec that is requested as an update exists in the prefix

**Parameters**

- **prefix** -- The target install prefix
- **specs** -- List of specs to be updated

**Raises**

- ***CondaError*** -- if there is an invalid spec provided
- ***PackageNotInstalledError*** -- if the requested specs to install don't exist in the prefix

**install_clone**(*args*, *parser*)

Executes an install of a new conda environment by cloning.

**install**(*args*, *parser*, *command='install'*)

Logic for *conda install*, *conda update*, *conda remove*, and *conda create*.

**install_revision**(*args*, *parser*)

Install a previous version of a conda environment

**revert_actions**(*prefix*, *revision=-1*, *index:* conda.core.index.Index | *None* = *None*)

**handle_txn**(*unlink_link_transaction*, *prefix*, *args*, *newenv*, *remove_op=False*)

### main

Entry point for all conda subcommands.

## Functions

| | |
|---|---|
| *init_loggers*() | |
| *main_subshell*(*args[, post_parse_hook]) | Entrypoint for the "subshell" invocation of CLI interface. E.g. *conda create*. |
| *main_sourced*(shell, *args, **kwargs) | Entrypoint for the "sourced" invocation of CLI interface. E.g. *conda activate*. |
| *main*(*args, **kwargs) | |

**init_loggers**()

**main_subshell**(**args*, *post_parse_hook=None*, ***kwargs*)

Entrypoint for the "subshell" invocation of CLI interface. E.g. *conda create*.

**main_sourced**(*shell*, **args*, ***kwargs*)

Entrypoint for the "sourced" invocation of CLI interface. E.g. *conda activate*.

**main**(**args*, ***kwargs*)

### main_clean

CLI implementation for *conda clean*.

Removes cached package tarballs, index files, package metadata, temporary files, and log files.

### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *_get_size*(→ int) |
| *_get_pkgs_dirs*(→ dict[str, tuple[str, Ellipsis]]) |
| *_get_total_size*(→ int) |
| *_rm_rf*(→ None) |
| *find_tarballs*(→ dict[str, Any]) |
| *find_pkgs*(→ dict[str, Any]) |
| *rm_pkgs*(→ None) |
| *find_index_cache*(→ list[str]) |
| *find_pkgs_dirs*(→ list[str]) |
| *find_tempfiles*(→ list[str]) |
| *find_logfiles*(→ list[str]) |
| *rm_items*(→ None) |
| *_execute*(args, parser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**_get_size**(**parts: str*, *warnings: list[str] | None*) → int

**_get_pkgs_dirs**(*pkg_sizes: dict[str, dict[str, int]]*) → dict[str, tuple[str, Ellipsis]]

**_get_total_size**(*pkg_sizes: dict[str, dict[str, int]]*) → int

**_rm_rf**(**parts: str*, *quiet: bool*, *verbose: bool*) → None

**find_tarballs**() → dict[str, Any]

**find_pkgs**() → dict[str, Any]

**rm_pkgs**(*pkgs_dirs: dict[str, tuple[str]]*, *warnings: list[str]*, *total_size: int*, *pkg_sizes: dict[str, dict[str, int]]*, ***, *quiet: bool*, *verbose: bool*, *dry_run: bool*, *name: str*) → None

**find_index_cache**() → list[str]

**find_pkgs_dirs**() → list[str]

**find_tempfiles**(*paths: collections.abc.Iterable[str]*) → list[str]

**find_logfiles**() → list[str]

**rm_items**(*items: list[str]*, *, *quiet: bool*, *verbose: bool*, *dry_run: bool*, *name: str*) → None

**_execute**(*args*, *parser*)

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_commands

Mock CLI implementation for *conda activate*.

A mock implementation of the activate shell command for better UX.

## Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_compare

CLI implementation for *conda compare*.

Compare the packages in an environment with the packages listed in an environment file.

## Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *get_packages*(prefix) |
| *compare_packages*(→ tuple[int, list[str]]) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**get_packages**(*prefix*)

**compare_packages**(*active_pkgs*, *specification_pkgs*) → tuple[int, list[str]]

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_config

CLI implementation for *conda config*.

Allows for programmatically interacting with conda's configuration files (e.g., *~/.condarc*).

## Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |
| *format_dict*(d) | |
| *parameter_description_builder*(name[, context, plugins]) | |
| *describe_all_parameters*(→ str) | Return a string with the descriptions of all available configuration |
| *print_config_item*(key, value) | |
| *_key_exists*(→ bool) | Logic to determine if the key is a valid setting. |
| *_get_key*(→ None) | |
| *_set_key*(→ None) | |
| *_remove_item*(→ None) | |
| *_remove_key*(→ None) | |
| *_read_rc*(→ dict) | |
| *_write_rc*(→ None) | |
| *_validate_provided_parameters*(→ None) | Compares the provided parameters with the available parameters. |
| *set_keys*(→ None) | |
| *execute_config*(args, parser) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

**format_dict**(*d*)

**parameter_description_builder**(*name*, *context=None*, *plugins=False*)

**describe_all_parameters**(*context=None*, *plugins=False*) → str

Return a string with the descriptions of all available configuration

When `context` has no parameters, this function returns ""

**print_config_item**(*key*, *value*)

**_key_exists**(*key: str*, *warnings: list[str]*, *context=None*) → bool

Logic to determine if the key is a valid setting.

**_get_key**(*key: str*, *config: dict*, *, *json: dict[str, Any] | None = None*, *warnings: list[str] | None = None*) → None

**_set_key**(*key: str*, *item: Any*, *config: dict*) → None

**_remove_item**(*key: str*, *item: Any*, *config: dict*) → None

**_remove_key**(*key: str*, *config: dict*) → None

**_read_rc**(*path: str | os.PathLike | pathlib.Path*) → dict

**_write_rc**(*path: str | os.PathLike | pathlib.Path*, *config: dict*) → None

**_validate_provided_parameters**(*parameters: collections.abc.Sequence[str]*, *plugin_parameters: collections.abc.Sequence[str]*, *context: conda.base.context.Context*) → None

Compares the provided parameters with the available parameters.

**Raises**

ArgumentError: If the provided parameters are not valid.

**set_keys**(**args: tuple[str, Any]*, *path: str | os.PathLike | pathlib.Path*) → None

**execute_config**(*args*, *parser*)

### main_create

CLI implementation for *conda create*.

Creates new conda environments with the specified packages.

## Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_env

Entry point for all conda-env subcommands.

#### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_env_config

CLI implementation for *conda-env config*.

Allows for programmatically interacting with conda-env's configuration files (e.g., *~/.condarc*).

#### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_env_create

CLI implementation for *conda-env create*.

Creates new conda environments with the specified packages.

### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

#### main_env_list

CLI implementation for *conda-env list*, now aliased to *conda info --envs*.

Lists available conda environments.

### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

#### main_env_remove

CLI implementation for *conda-env remove*.

Removes the specified conda environment.

### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_env_update

CLI implementation for *conda-env update*.

Updates the conda environments with the specified packages.

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_env_vars

CLI implementation for *conda-env config vars*.

Allows for configuring conda-env's vars.

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute_list*(→ int) | |
| *execute_set*(→ int) | |
| *execute_unset*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute_list**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

**execute_set**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

**execute_unset**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_export

CLI implementation for *conda export*.

Dumps specified environment package specifications to the screen.

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_info

CLI implementation for *conda info*.

Display information about current conda installation.

#### Classes

| | |
|---|---|
| *InfoRenderer* | Provides a `render` method for rendering `InfoComponents` |

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *get_user_site*(→ list[str]) | Method used to populate `site_dirs` in `conda info`. |
| *dump_record*(→ dict[str, Any]) | Returns a dictionary of key/value pairs from `prec`. Keys included in `IGNORE_FIELDS` are not returned. |
| *pretty_package*(→ None) | Pretty prints contents of a `PackageRecord` |
| *get_info_dict*(→ dict[str, Any]) | Returns a dictionary of contextual information. |
| *get_env_vars_str*(→ str) | Returns a printable string representing environment variables from the dictionary returned by `get_info_dict`. |
| *get_main_info_display*(→ dict[str, str]) | Returns the data that can be used to display information for conda info |
| *get_main_info_str*(→ str) | Returns a printable string of the contents of `info_dict`. |
| *get_info_components*(→ set[InfoComponents]) | |
| *iter_info_components*(...) | Determine which components to display. |
| *execute*(→ int) | Implements `conda info` command. |

### Attributes

| |
|---|
| *IGNORE_FIELDS* |
| *SKIP_FIELDS* |
| *InfoComponents* |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**get_user_site**() → list[str]

Method used to populate `site_dirs` in `conda info`.

**Returns**
List of directories.

**IGNORE_FIELDS: set[str]**

**SKIP_FIELDS: set[str]**

**dump_record**(*prec:* conda.models.records.PackageRecord) → dict[str, Any]

Returns a dictionary of key/value pairs from `prec`. Keys included in `IGNORE_FIELDS` are not returned.

**Parameters**
**prec** -- A `PackageRecord` object.

**Returns**
A dictionary of elements dumped from `prec`

**pretty_package**(*prec:* conda.models.records.PackageRecord) → None

Pretty prints contents of a `PackageRecord`

**Parameters**
**prec** -- A `PackageRecord`

**get_info_dict**() → dict[str, Any]

Returns a dictionary of contextual information.

**Returns**
Dictionary of conda information to be sent to stdout.

**get_env_vars_str**(*info_dict: dict[str, Any]*) → str

Returns a printable string representing environment variables from the dictionary returned by `get_info_dict`.

**Parameters**
**info_dict** -- The returned dictionary from `get_info_dict()`.

**Returns**
String to print.

**get_main_info_display**(*info_dict: dict[str, Any]*) → dict[str, str]

Returns the data that can be used to display information for conda info

**get_main_info_str**(*info_dict: dict[str, Any]*) → str

Returns a printable string of the contents of `info_dict`.

**Parameters**
**info_dict** -- The output of `get_info_dict()`.

**Returns**
String to print.

`InfoComponents`

`class InfoRenderer`(*context*)

Provides a `render` method for rendering `InfoComponents`

`render`(*components: collections.abc.Iterable[InfoComponents]*)

Iterates through the registered components, obtains the data to render via a `_<component>_component` method and then renders it.

`_base_component`() → str | dict

`_channels_component`() → str | dict

`_detail_component`() → dict[str, str]

`_envs_component`()

`_system_component`() → str

`_json_all_component`() → dict[str, Any]

`get_info_components`(*args: argparse.Namespace*, *context:* conda.base.context.Context) → set[InfoComponents]

`iter_info_components`(*args: argparse.Namespace*, *context:* conda.base.context.Context) → collections.abc.Iterable[InfoComponents]

Determine which components to display.

**Parameters**

- **args** -- The parsed command line arguments.
- **context** -- The conda context.

**Returns**
An iterable of components to display.

`execute`(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Implements `conda info` command.

- `conda info`
- `conda info --base`
- `conda info <package_spec> ...`
- `conda info --unsafe-channels`
- `conda info --envs`
- `conda info --system`

### main_init

CLI implementation for *conda init*.

Prepares the user's profile for running conda, and sets up the conda shell interface.

#### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_install

CLI implementation for *conda install*.

Installs the specified packages into an existing environment.

#### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_list

CLI implementation for *conda list*.

Lists all packages installed into an environment.

### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *print_export_header*(subdir) | |
| *get_packages*(installed, regex) | |
| *list_packages*(→ tuple[int, list[str] \| list[dict[str, ...) | |
| *print_packages*(→ int) | |
| *print_explicit*(prefix[, add_md5, remove_auth, add_sha256]) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**print_export_header**(*subdir*)

**get_packages**(*installed*, *regex*)

**list_packages**(*prefix*, *regex=None*, *format='human'*, *reverse=False*, *show_channel_urls=None*, *reload_records=True*, *fields=None*) → tuple[int, list[str] | list[dict[str, Any]]]

**print_packages**(*prefix*, *regex=None*, *format='human'*, *reverse=False*, *piplist=False*, *json=False*, *show_channel_urls=None*, *fields=None*) → int

**print_explicit**(*prefix*, *add_md5=False*, *remove_auth=True*, *add_sha256=False*)

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

## main_mock_activate

Mock CLI implementation for *conda activate*.

A mock implementation of the activate shell command for better UX.

### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_mock_deactivate

Mock CLI implementation for *conda deactivate*.

A mock implementation of the deactivate shell command for better UX.

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_notices

CLI implementation for *conda notices*.

Manually retrieves channel notifications, caches them and displays them.

#### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *execute*(→ int) | Command that retrieves channel notifications, caches them and displays them. |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Command that retrieves channel notifications, caches them and displays them.

### main_package

CLI implementation for *conda package*.

Provides some low-level tools for creating conda packages.

### Functions

| | |
|---|---|
| *configure_parser*(→ argparse.ArgumentParser) | |
| *remove*(prefix, files) | Remove files for a given prefix. |
| *execute*(→ int) | |
| *get_installed_version*(prefix, name) | |
| *create_info*(name, version, build_number, requires_py) | |
| *fix_shebang*(tmp_dir, path) | |
| *_add_info_dir*(t, tmp_dir, files, has_prefix, info) | |
| *create_conda_pkg*(prefix, files, info, tar_path[, ...]) | Create a conda package and return a list of warnings. |
| *make_tarbz2*(prefix[, name, version, build_number, files]) | |
| *which_package*(path) | Return the package containing the path. |
| *which_prefix*(path) | Return the prefix for the provided path. |

### Attributes

| |
|---|
| *shebang_pat* |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**remove**(*prefix*, *files*)

Remove files for a given prefix.

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

**get_installed_version**(*prefix*, *name*)

**create_info**(*name*, *version*, *build_number*, *requires_py*)

**shebang_pat**

**fix_shebang**(*tmp_dir*, *path*)

**_add_info_dir**(*t*, *tmp_dir*, *files*, *has_prefix*, *info*)

**create_conda_pkg**(*prefix*, *files*, *info*, *tar_path*, *update_info=None*)

Create a conda package and return a list of warnings.

**make_tarbz2**(*prefix*, *name='unknown'*, *version='0.0'*, *build_number=0*, *files=None*)

**which_package**(*path*)

Return the package containing the path.

Provided the path of a (presumably) conda installed file, iterate over the conda packages the file came from. Usually the iteration yields only one package.

**which_prefix**(*path*)

Return the prefix for the provided path.

Provided the path of a (presumably) conda installed file, return the environment prefix in which the file in located.

### main_pip

PEP 621 compatible entry point used when *conda init* has not updated the user shell profile.

#### Functions

| |
|---|
| *pip_installed_post_parse_hook*(args, p) |
| *main*(*args, **kwargs) |

**pip_installed_post_parse_hook**(*args*, *p*)

**main**(**args*, ***kwargs*)

### main_remove

CLI implementation for *conda remove*.

Removes the specified packages from an existing environment.

#### Functions

| |
|---|
| *configure_parser*(→ argparse.ArgumentParser) |
| *execute*(→ int) |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_rename

CLI implementation for *conda rename*.

Renames an existing environment by cloning it and then removing the original environment.

### Functions

| | |
|---|---|
| configure_parser(→ argparse.ArgumentParser) | |
| check_protected_dirs(→ None) | Ensure that the new prefix does not contain protected directories. |
| validate_src(→ str) | Ensure that we are receiving at least one valid value for the environment |
| validate_destination(→ str) | Ensure that our destination does not exist |
| execute(→ int) | Executes the command for renaming an existing environment. |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**check_protected_dirs**(*prefix: str | pathlib.Path*, *json: bool = False*) → None

Ensure that the new prefix does not contain protected directories.

**validate_src**() → str

Ensure that we are receiving at least one valid value for the environment to be renamed and that the "base" environment is not being renamed

**validate_destination**(*dest: str*, *force: bool = False*) → str

Ensure that our destination does not exist

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Executes the command for renaming an existing environment.

## main_run

CLI implementation for *conda run*.

Runs the provided command within the specified environment.

### Functions

| | |
|---|---|
| configure_parser(→ argparse.ArgumentParser) | |
| execute(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### main_search

CLI implementation for *conda search*.

Query channels for packages matching the provided package spec.

#### Functions

| | |
|---|---|
| configure_parser(→ argparse.ArgumentParser) | |
| execute(→ int) | Implements *conda search* commands. |
| _pretty_record_format(→ str) | Format a *PackageRecord* for *pretty_record()* |
| pretty_record(→ None) | Pretty prints a *PackageRecord*. |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

Implements *conda search* commands.

*conda search <spec>* searches channels for packages. *conda search <spec> --envs* searches environments for packages.

**_pretty_record_format**(*record:* conda.models.records.PackageRecord) → str

Format a *PackageRecord* for *pretty_record()*

**pretty_record**(*record:* conda.models.records.PackageRecord, *print=print*) → None

Pretty prints a *PackageRecord*.

**Parameters**
**record** -- The *PackageRecord* object to print.

### main_update

CLI implementation for *conda update*.

Updates the specified packages in an existing environment.

#### Functions

| | |
|---|---|
| configure_parser(→ argparse.ArgumentParser) | |
| execute(→ int) | |

**configure_parser**(*sub_parsers: argparse._SubParsersAction*, ***kwargs*) → argparse.ArgumentParser

**execute**(*args: argparse.Namespace*, *parser: argparse.ArgumentParser*) → int

### python_api

Wrapper for running conda CLI commands as a Python API.

## Classes

| | |
|---|---|
| *Commands* | |

## Functions

| | |
|---|---|
| *run_command*(command, *arguments, **kwargs) | Runs a conda command in-process with a given set of command-line interface arguments. |

## Attributes

| | |
|---|---|
| *STRING* | |
| *STDOUT* | |

class Commands

CLEAN = 'clean'

CONFIG = 'config'

CREATE = 'create'

INFO = 'info'

INSTALL = 'install'

LIST = 'list'

REMOVE = 'remove'

SEARCH = 'search'

UPDATE = 'update'

RUN = 'run'

NOTICES = 'notices'

STRING

STDOUT

**run_command**(*command*, **arguments*, ***kwargs*)

Runs a conda command in-process with a given set of command-line interface arguments.

**Differences from the command-line interface:**

Always uses --yes flag, thus does not ask for confirmation.

**Parameters**

- **command** -- one of the Commands.
- ***arguments** -- instructions you would normally pass to the conda command on the command line see below for examples. Be very careful to delimit arguments exactly as you want them to be delivered. No 'combine then split at spaces' or other information destroying processing gets performed on the arguments.
- ****kwargs** -- special instructions for programmatic overrides

**Keyword Arguments**

- **use_exception_handler** -- defaults to False. False will let the code calling *run_command* handle all exceptions. True won't raise when an exception has occurred, and instead give a non-zero return code
- **search_path** -- an optional non-standard search path for configuration information that overrides the default SEARCH_PATH
- **stdout** -- Define capture behavior for stream sys.stdout. Defaults to STRING. STRING captures as a string. None leaves stream untouched. Otherwise redirect to file-like object stdout.
- **stderr** -- Define capture behavior for stream sys.stderr. Defaults to STRING. STRING captures as a string. None leaves stream untouched. STDOUT redirects to stdout target and returns None as stderr value. Otherwise redirect to file-like object stderr.

**Returns**

a tuple of stdout, stderr, and return_code. stdout, stderr are either strings, None or the corresponding file-like function argument.

### Examples

```
>>> run_command(Commands.CREATE, "-n", "newenv", "python=3", "flask",
↪          use_exception_handler=True)
>>> run_command(Commands.CREATE, "-n", "newenv", "python=3", "flask")
>>> run_command(Commands.CREATE, ["-n", "newenv", "python=3", "flask"], search_
↪path=())
```

## Functions

*main*(*args, **kwargs)

**main**(**args*, ***kwargs*)

### common

Code in `conda.common` is not conda-specific. Technically, it sits *aside* the application stack and not *within* the stack. It is able to stand independently on its own. The *only* allowed imports of conda code in `conda.common` modules are imports of other `conda.common` modules.

If objects are needed from other parts of conda, they should be passed directly as arguments to functions and methods.

### _logic

## Classes

| | |
|---|---|
| *_ClauseList* | Storage for the CNF clauses, represented as a list of tuples of ints. |
| *_ClauseArray* | Storage for the CNF clauses, represented as a flat int array. |
| *_SatSolver* | Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance. |
| *_PycoSatSolver* | Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance. |
| *_PyCryptoSatSolver* | Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance. |
| *_PySatSolver* | Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance. |
| *Clauses* | |

## Attributes

| | |
|---|---|
| *TRUE* | |
| *FALSE* | |
| *_sat_solver_str_to_cls* | |
| *_sat_solver_cls_to_str* | |

**TRUE**

**FALSE**

**class _ClauseList**

Storage for the CNF clauses, represented as a list of tuples of ints.

**get_clause_count()**

Return number of stored clauses.

**save_state**()

Get state information to be able to revert temporary additions of supplementary clauses. _ClauseList: state is simply the number of clauses.

**restore_state**(*saved_state*)

Restore state saved via *save_state*. Removes clauses that were added after the state has been saved.

**as_list**()

Return clauses as a list of tuples of ints.

**as_array**()

Return clauses as a flat int array, each clause being terminated by 0.

**class _ClauseArray**

Storage for the CNF clauses, represented as a flat int array. Each clause is terminated by int(0).

**extend**(*clauses*)

**append**(*clause*)

**get_clause_count**()

Return number of stored clauses. This is an O(n) operation since we don't store the number of clauses explicitly due to performance reasons (Python interpreter overhead in self.append).

**save_state**()

Get state information to be able to revert temporary additions of supplementary clauses. _ClauseArray: state is the length of the int array, NOT number of clauses.

**restore_state**(*saved_state*)

Restore state saved via *save_state*. Removes clauses that were added after the state has been saved.

**as_list**()

Return clauses as a list of tuples of ints.

**as_array**()

Return clauses as a flat int array, each clause being terminated by 0.

**class _SatSolver**(***run_kwargs*)

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

**get_clause_count**()

**as_list**()

**save_state**()

**restore_state**(*saved_state*)

**run**(*m*, ***kwargs*)

**abstract setup**(*m*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

**abstract invoke**(*solver*)

Start the actual SAT solving and return the calculated solution.

**abstract process_solution**(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PycoSatSolver(**run_kwargs)

Bases: _SatSolver

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, *limit=0*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*iter_sol*)

Start the actual SAT solving and return the calculated solution.

process_solution(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PyCryptoSatSolver(**run_kwargs)

Bases: _SatSolver

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, *threads=1*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*solver*)

Start the actual SAT solving and return the calculated solution.

process_solution(*solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

class _PySatSolver(**run_kwargs)

Bases: _SatSolver

Simple wrapper to call a SAT solver given a _ClauseList/_ClauseArray instance.

setup(*m*, ***kwargs*)

Create a solver instance, add the clauses to it, and return it.

invoke(*solver*)

Start the actual SAT solving and return the calculated solution.

process_solution(*sat_solution*)

Process the solution returned by self.invoke. Returns a list of satisfied variables or None if no solution is found.

_sat_solver_str_to_cls

_sat_solver_cls_to_str

class Clauses(*m=0*, *sat_solver_str=_sat_solver_cls_to_str[_PycoSatSolver]*)

get_clause_count()

as_list()

new_var()

assign(*vals*)

Combine(*args*, *polarity*)

**Eval**(*func*, *args*, *polarity*)

**Prevent**(*func*, **args*)

**Require**(*func*, **args*)

**Not**(*x*, *polarity=None*, *add_new_clauses=False*)

**And**(*f*, *g*, *polarity*, *add_new_clauses=False*)

**Or**(*f*, *g*, *polarity*, *add_new_clauses=False*)

**Xor**(*f*, *g*, *polarity*, *add_new_clauses=False*)

**ITE**(*c*, *t*, *f*, *polarity*, *add_new_clauses=False*)

**All**(*iter*, *polarity=None*)

**Any**(*iter*, *polarity*)

**AtMostOne_NSQ**(*vals*, *polarity*)

**AtMostOne_BDD**(*vals*, *polarity=None*)

**ExactlyOne_NSQ**(*vals*, *polarity*)

**ExactlyOne_BDD**(*vals*, *polarity*)

**LB_Preprocess**(*lits*, *coeffs*)

**BDD**(*lits*, *coeffs*, *nterms*, *lo*, *hi*, *polarity*)

**LinearBound**(*lits*, *coeffs*, *lo*, *hi*, *preprocess*, *polarity*)

**_run_sat**(*m*, *limit=0*)

**sat**(*additional=None*, *includeIf=False*, *limit=0*)

Calculate a SAT solution for the current clause set.

Returned is the list of those solutions. When the clauses are unsatisfiable, an empty list is returned.

**minimize**(*lits*, *coeffs*, *bestsol=None*, *trymax=False*)

Minimize the objective function given by (coeff, integer) pairs in zip(coeffs, lits). The actual minimization is multiobjective: first, we minimize the largest active coefficient value, then we minimize the sum.

### _os

### linux

## Functions

| | |
|---|---|
| *linux_get_libc_version*(→ tuple[str, str] \| tuple[None, ...) | If on linux, returns (libc_family, version), otherwise (None, None). |

**linux_get_libc_version**() → tuple[str, str] | tuple[None, None]

If on linux, returns (libc_family, version), otherwise (None, None).

### osx

#### Functions

| | |
|---|---|
| *mac_ver*(→ str) | Returns macOS version, without compatibility modes for 11.x. |

**mac_ver**() → str

Returns macOS version, without compatibility modes for 11.x. https://github.com/conda/conda/issues/13832 If Python was compiled against macOS <=10.15, we might get 10.16 instead of 11.0. For these cases, we must set SYSTEM_VERSION_COMPAT=0 and call sw_vers directly.

### unix

#### Functions

| | |
|---|---|
| *get_free_space_on_unix*(dir_name) | |
| *is_admin_on_unix*() | |

**get_free_space_on_unix**(*dir_name*)

**is_admin_on_unix**()

### windows

#### Classes

| | |
|---|---|
| *SW* | Enum where members are also (and must be) ints |
| *ERROR* | Enum where members are also (and must be) ints |

#### Functions

| | |
|---|---|
| *get_free_space_on_windows*(dir_name) | |
| *is_admin_on_windows*() | |
| *_wait_and_close_handle*(process_handle) | Waits until spawned process finishes and closes the handle for it. |
| *run_as_admin*(args[, wait]) | Run command line argument list (*args*) with elevated privileges. |

### Attributes

*PHANDLE*

**PHANDLE**

**class SW**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Initialize self. See help(type(self)) for accurate signature.

**HIDE = 0**

**MAXIMIZE = 3**

**MINIMIZE = 6**

**RESTORE = 9**

**SHOW = 5**

**SHOWDEFAULT = 10**

**SHOWMAXIMIZED = 3**

**SHOWMINIMIZED = 2**

**SHOWMINNOACTIVE = 7**

**SHOWNA = 8**

**SHOWNOACTIVATE = 4**

**SHOWNORMAL = 1**

**class ERROR**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Initialize self. See help(type(self)) for accurate signature.

**ZERO = 0**

**FILE_NOT_FOUND = 2**

**PATH_NOT_FOUND = 3**

**BAD_FORMAT = 11**

**ACCESS_DENIED = 5**

**ASSOC_INCOMPLETE = 27**

**DDE_BUSY = 30**

DDE_FAIL = 29

DDE_TIMEOUT = 28

DLL_NOT_FOUND = 32

NO_ASSOC = 31

OOM = 8

SHARE = 26

get_free_space_on_windows(*dir_name*)

is_admin_on_windows()

_wait_and_close_handle(*process_handle*)

Waits until spawned process finishes and closes the handle for it.

run_as_admin(*args*, *wait=True*)

Run command line argument list (*args*) with elevated privileges.

If *wait* is True, the process will block until completion.

#### Notes

- no stdin / stdout / stderr pipe support
- does not automatically quote arguments (i.e. for paths that may contain spaces)

See: - http://stackoverflow.com/a/19719292/1170370 on 20160407 MCS. - msdn.microsoft.com/en-us/library/windows/desktop/bb762153(v=vs.85).aspx - https://github.com/ContinuumIO/menuinst/blob/master/menuinst/windows/win_elevate.py - https://github.com/saltstack/salt-windows-install/blob/master/deps/salt/python/App/Lib/site-packages/win32/Demos/pipes/runproc.py # NOQA - https://github.com/twonds/twisted/blob/master/twisted/internet/_dumbwin32proc.py - https://stackoverflow.com/a/19982092/2127762 - https://www.codeproject.com/Articles/19165/Vista-UAC-The-Definitive-Guide - https://github.com/JustAMan/pyWinClobber/blob/master/win32elevate.py

on_win

### compat

Common compatibility code.

### Functions

| | |
|---|---|
| *encode_for_env_var*(→ str) | Environment names and values need to be string. |
| *encode_environment*(env) | |
| *isiterable*(obj) | |
| *open_utf8*(file[, mode, buffering, encoding, errors, ...]) | |
| *open*(file[, mode, buffering, encoding, errors, ...]) | |
| *six_with_metaclass*(meta, *bases) | Create a base class with a metaclass. |
| *ensure_binary*(value) | |
| *ensure_text_type*(→ str) | |
| *ensure_unicode*(value) | |
| *ensure_fs_path_encoding*(value) | |
| *ensure_utf8_encoding*(value) | |

### Attributes

| |
|---|
| *on_win* |
| *on_mac* |
| *on_linux* |
| *ENCODE_ENVIRONMENT* |
| *NoneType* |
| *primitive_types* |

**on_win**

**on_mac**

**on_linux**

**ENCODE_ENVIRONMENT = True**

**encode_for_env_var**(*value*) → str

Environment names and values need to be string.

**encode_environment**(*env*)

isiterable(*obj*)

open_utf8(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*)

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*)

six_with_metaclass(*meta*, **bases*)

Create a base class with a metaclass.

NoneType

primitive_types = ()

ensure_binary(*value*)

ensure_text_type(*value*) → str

ensure_unicode(*value*)

ensure_fs_path_encoding(*value*)

ensure_utf8_encoding(*value*)

### configuration

A generalized application configuration utility.

**Features include:**

- lazy eval
- merges configuration files
- parameter type validation, with custom validation
- parameter aliases

Easily extensible to other source formats, e.g. json and ini

### Classes

| | |
|---|---|
| *ParameterFlag* | Create a collection of name/value pairs. |
| *RawParameter* | |
| *EnvRawParameter* | |
| *ArgParseRawParameter* | |
| *YamlRawParameter* | |
| *DefaultValueRawParameter* | Wraps a default value as a RawParameter, for usage in ParameterLoader. |
| *LoadedParameter* | Represents a Parameter that has been loaded with configuration value. |
| *PrimitiveLoadedParameter* | LoadedParameter type that holds a single python primitive value. |
| *MapLoadedParameter* | LoadedParameter type that holds a map (i.e. dict) of LoadedParameters. |
| *SequenceLoadedParameter* | LoadedParameter type that holds a sequence (i.e. list) of LoadedParameters. |
| *ObjectLoadedParameter* | LoadedParameter type that holds a mapping (i.e. object) of LoadedParameters. |
| *ConfigurationObject* | Dummy class to mark whether a Python object has config parameters within. |
| *Parameter* | The Parameter class represents an unloaded configuration parameter, holding type, default |
| *PrimitiveParameter* | Parameter type for a Configuration class that holds a single python primitive value. |
| *MapParameter* | Parameter type for a Configuration class that holds a map (i.e. dict) of Parameters. |
| *SequenceParameter* | Parameter type for a Configuration class that holds a sequence (i.e. list) of Parameters. |
| *ObjectParameter* | Parameter type for a Configuration class that holds an object with Parameter fields. |
| *ParameterLoader* | ParameterLoader class contains the top level logic needed to load a parameter from start to |
| *ConfigurationType* | metaclass for Configuration |
| *Configuration* | |

### Functions

| | |
|---|---|
| *pretty_list*(iterable[, padding]) | |
| *pretty_map*(dictionary[, padding]) | |
| *expand_environment_variables*(unexpanded) | |
| *raise_errors*(errors) | |
| *custom_expandvars*(→ str) | Expand variables in a string. |
| *unique_sequence_map*(*, unique_key) | Used to validate properties on *Configuration* subclasses defined as a |

### Attributes

| |
|---|
| *EMPTY_MAP* |
| *CONDARC_FILENAMES* |
| *YAML_EXTENSIONS* |
| *_RE_CUSTOM_EXPANDVARS* |

**EMPTY_MAP**

**pretty_list**(*iterable*, *padding=' '*)

**pretty_map**(*dictionary*, *padding=' '*)

**expand_environment_variables**(*unexpanded*)

**exception ConfigurationError**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ConfigurationLoadError**(*path*, *message_addition=''*, ***kwargs*)

Bases: *ConfigurationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ValidationError**(*parameter_name*, *parameter_value*, *source*, *msg=None*, ***kwargs*)

Bases: *ConfigurationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception MultipleKeysError**(*source*, *keys*, *preferred_key*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception InvalidTypeError**(*parameter_name*, *parameter_value*, *source*, *wrong_type*, *valid_types*, *msg=None*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CustomValidationError**(*parameter_name*, *parameter_value*, *source*, *custom_message*)

Bases: *ValidationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception MultiValidationError**(*errors*, **args*, ***kwargs*)

Bases: *conda.CondaMultiError*, *ConfigurationError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**raise_errors**(*errors*)

**class ParameterFlag**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**final = 'final'**

**top = 'top'**

**bottom = 'bottom'**

**__str__**()

Return str(self).

**classmethod from_name**(*name*)

**classmethod from_value**(*value*)

**classmethod from_string**(*string*)

**class RawParameter**(*source*, *key*, *raw_value*)

**__repr__**()

Return repr(self).

**abstract value**(*parameter_obj*)

**abstract keyflag**()

**abstract valueflags**(*parameter_obj*)

**classmethod make_raw_parameters**(*source*, *from_map*)

**class EnvRawParameter**(*source*, *key*, *raw_value*)

Bases: *RawParameter*

**property __important_split_value**

**source = 'envvars'**

**value**(*parameter_obj*)

**keyflag**()

**valueflags**(*parameter_obj*)

**classmethod make_raw_parameters**(*appname*)

**class ArgParseRawParameter**(*source*, *key*, *raw_value*)

Bases: *RawParameter*

**source = 'cmd_line'**

**value**(*parameter_obj*)

**keyflag**()

**valueflags**(*parameter_obj*)

**classmethod make_raw_parameters**(*args_from_argparse*)

**class YamlRawParameter**(*source*, *key*, *raw_value*, *key_comment*)

Bases: RawParameter

**value**(*parameter_obj*)

**keyflag**()

**valueflags**(*parameter_obj*)

**static _get_yaml_key_comment**(*commented_dict*, *key*)

**classmethod _get_yaml_list_comments**(*value*)

**static _get_yaml_list_comment_item**(*item*)

**static _get_yaml_map_comments**(*value*)

**classmethod make_raw_parameters**(*source*, *from_map*)

**classmethod make_raw_parameters_from_file**(*filepath*)

**class DefaultValueRawParameter**(*source*, *key*, *raw_value*)

Bases: RawParameter

Wraps a default value as a RawParameter, for usage in ParameterLoader.

**value**(*parameter_obj*)

**keyflag**()

**valueflags**(*parameter_obj*)

**class LoadedParameter**(*name*, *value*, *key_flag*, *value_flags*, *validation=None*)

Represents a Parameter that has been loaded with configuration value.

**Parameters**

- **name** (*str*) -- name of the loaded parameter
- **value** (*LoadedParameter or primitive*) -- the value of the loaded parameter
- **key_flag** (*ParameterFlag or None*) -- priority flag for the parameter itself
- **value_flags** (*Any or None*) -- priority flags for the parameter values
- **validation** (*callable*) -- Given a parameter value as input, return a boolean indicating validity, or alternately return a string describing an invalid value.

**_type**

**_element_type**

**__eq__**(*other*)

Return self==value.

__hash__()

Return hash(self).

collect_errors(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

**Parameters**

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

expand()

Recursively expands any environment values in the Loaded Parameter.

Returns LoadedParameter

abstract merge(*matches*)

Recursively merges matches into one LoadedParameter.

**Parameters**
**matches** (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

typify(*source*)

Recursively types a LoadedParameter.

**Parameters**
**source** (*str*) -- string describing the source of the LoadedParameter.

Returns: a primitive, sequence, or map representing the typed value.

static _typify_data_structure(*value*, *source*, *type_hint=None*)

static _match_key_is_important(*loaded_parameter*)

static _first_important_matches(*matches*)

class PrimitiveLoadedParameter(*name*, *element_type*, *value*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a single python primitive value.

The python primitive types are str, int, float, complex, bool, and NoneType. In addition, python 2 has long and unicode types.

**Parameters**

- **element_type** (*type or tuple[type]*) -- Type-validation of parameter's value.
- **value** (*primitive value*) -- primitive python value.

__eq__(*other*)

Return self==value.

__hash__()

Return hash(self).

**merge**(*matches*)

Recursively merges matches into one LoadedParameter.

**Parameters**
**matches** (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

**class MapLoadedParameter**(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a map (i.e. dict) of LoadedParameters.

**Parameters**

- **value** (*Mapping*) -- Map of string keys to LoadedParameter values.
- **element_type** (Parameter) -- The Parameter type that is held in value.
- **value_flags** (*Mapping*) -- Map of priority value flags.

**_type**

**collect_errors**(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

**Parameters**

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

**merge**(*parameters: collections.abc.Sequence[*MapLoadedParameter*]*) → *MapLoadedParameter*

Recursively merges matches into one LoadedParameter.

**Parameters**
**matches** (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

**class SequenceLoadedParameter**(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a sequence (i.e. list) of LoadedParameters.

**Parameters**

- **value** (*Sequence*) -- Sequence of LoadedParameter values.
- **element_type** (Parameter) -- The Parameter type that is held in the sequence.
- **value_flags** (*Sequence*) -- Sequence of priority value_flags.

**_type**

**collect_errors**(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

**Parameters**

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

**merge**(*matches*)

Recursively merges matches into one LoadedParameter.

**Parameters**
**matches** (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

**class ObjectLoadedParameter**(*name*, *value*, *element_type*, *key_flag*, *value_flags*, *validation=None*)

Bases: *LoadedParameter*

LoadedParameter type that holds a mapping (i.e. object) of LoadedParameters.

**Parameters**

- **value** (*Sequence*) -- Object with LoadedParameter fields.
- **element_type** (*object*) -- The Parameter type that is held in the sequence.
- **value_flags** (*Sequence*) -- Sequence of priority value_flags.

**_type**

**collect_errors**(*instance*, *typed_value*, *source='<<merged>>'*)

Validate a LoadedParameter typed value.

**Parameters**

- **instance** (Configuration) -- the instance object used to create the LoadedParameter.
- **typed_value** (*Any*) -- typed value to validate.
- **source** (*str*) -- string description for the source of the typed_value.

**merge**(*parameters: collections.abc.Sequence[*ObjectLoadedParameter*]*) → *ObjectLoadedParameter*

Recursively merges matches into one LoadedParameter.

**Parameters**
**matches** (*List<LoadedParameter>*) -- list of matches of this parameter.

Returns: LoadedParameter

**class ConfigurationObject**

Dummy class to mark whether a Python object has config parameters within.

**to_json**()

Return a serializable object with defaults filled in

**class Parameter**(*default*, *validation=None*)

The Parameter class represents an unloaded configuration parameter, holding type, default and validation information until the parameter is loaded with a configuration.

**Parameters**

- **default** (*Any*) -- the typed, python representation default value given if the Parameter is not found in a Configuration.
- **validation** (*callable*) -- Given a parameter value as input, return a boolean indicating validity, or alternately return a string describing an invalid value.

**property default**

Returns a DefaultValueRawParameter that wraps the actual default value.

**_type**

**_element_type**

**get_all_matches**(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

**Parameters**

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

**abstract load**(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

**Parameters**

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

**typify**(*name*, *source*, *value*)

**class PrimitiveParameter**(*default*, *element_type=None*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a single python primitive value.

The python primitive types are str, int, float, complex, bool, and NoneType. In addition, python 2 has long and unicode types.

**Parameters**

- **default** (*primitive value*) -- default value if the Parameter is not found.
- **element_type** (*type or tuple[type]*) -- Type-validation of parameter's value. If None, type(default) is used.

**load**(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

**Parameters**

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

**class MapParameter**(*element_type*, *default=frozendict()*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a map (i.e. dict) of Parameters.

**Parameters**

- **element_type** (Parameter) -- The Parameter type held in the MapParameter.
- **default** (*Mapping*) -- The parameter's default value. If None, will be an empty dict.

_type

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

**Parameters**

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

**Parameters**

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class SequenceParameter(*element_type*, *default=()*, *validation=None*, *string_delimiter=','*)

Bases: *Parameter*

Parameter type for a Configuration class that holds a sequence (i.e. list) of Parameters.

**Parameters**

- **element_type** (Parameter) -- The Parameter type that is held in the sequence.
- **default** (*Sequence*) -- default value, empty tuple if not given.
- **string_delimiter** (*str*) -- separation string used to parse string into sequence.

_type

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

**Parameters**

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

**Parameters**

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class ObjectParameter(*element_type*, *default=ConfigurationObject()*, *validation=None*)

Bases: *Parameter*

Parameter type for a Configuration class that holds an object with Parameter fields.

**Parameters**

- **element_type** (*object*) -- The object type with parameter fields held in ObjectParameter.
- **default** (*Sequence*) -- default value, empty tuple if not given.

_type

get_all_matches(*name*, *names*, *instance*)

Finds all matches of a Parameter in a Configuration instance

**Parameters**

- **name** (*str*) -- canonical name of the parameter to search for
- **names** (*tuple(str)*) -- alternative aliases of the parameter
- **instance** (Configuration) -- instance of the configuration to search within

Returns (List(RawParameter)): matches of the parameter found in the configuration.

load(*name*, *match*)

Loads a Parameter with the value in a RawParameter.

**Parameters**

- **name** (*str*) -- name of the parameter to pass through
- **match** (RawParameter) -- the value of the RawParameter match

Returns a LoadedParameter

class ParameterLoader(*parameter_type*, *aliases=()*, *expandvars=False*)

ParameterLoader class contains the top level logic needed to load a parameter from start to finish.

**Parameters**

- **parameter_type** (Parameter) -- the type of Parameter that is stored in the loader.
- **aliases** (*tuple(str)*) -- alternative aliases for the Parameter
- **expandvars** (*bool*) -- whether or not to recursively expand environmental variables.

property name

property names

_set_name(*name*)

__get__(*instance*, *instance_type*)

_raw_parameters_from_single_source(*raw_parameters*)

static raw_parameters_from_single_source(*name*, *names*, *raw_parameters*)

class ConfigurationType(*name*, *bases*, *attr*)

Bases: type

metaclass for Configuration

property _parameter_loaders: dict[str, ParameterLoader]

__call__(**args*, ***kwargs*)

Call self as a function.

CONDARC_FILENAMES = ('.condarc', 'condarc')

YAML_EXTENSIONS = ('.yml', '.yaml')

_RE_CUSTOM_EXPANDVARS

custom_expandvars(*template: str*, *mapping: collections.abc.Mapping[str, Any] = {}*, */*, ***kwargs*) → str

Expand variables in a string.

Inspired by *string.Template* and modified to mirror *os.path.expandvars* functionality allowing custom variables without mutating *os.environ*.

Expands POSIX and Windows CMD environment variables as follows:

- $VARIABLE → value of VARIABLE
- ${VARIABLE} → value of VARIABLE
- %VARIABLE% → value of VARIABLE

Invalid substitutions are left as-is:

- $MISSING → $MISSING
- ${MISSING} → ${MISSING}
- %MISSING% → %MISSING%
- $$ → $$
- %% → %%
- $ → $
- % → %

class Configuration(*search_path=()*, *app_name=None*, *argparse_args=None*, ***kwargs*)

classmethod _set_parameter_names_and_aliases()

Build parameter_names_and_aliases from the class's parameter loaders.

static _expand_search_path(*search_path: conda.common.path.PathsType*, ***kwargs*) → collections.abc.Iterable[pathlib.Path]

classmethod _load_search_path(*search_path: collections.abc.Iterable[pathlib.Path]*) → collections.abc.Iterable[tuple[pathlib.Path, dict]]

_set_search_path(*search_path: conda.common.path.PathsType*, ***kwargs*)

_set_env_vars(*app_name=None*)

_set_argparse_args(*argparse_args*)

_set_raw_data(*raw_data: collections.abc.Mapping[collections.abc.Hashable, dict]*)

**name_for_alias**(*alias: str*, *ignore_private: bool = True*) → str | None

Find the canonical parameter name for a given alias.

This method searches through all configuration parameters to find the canonical parameter name that corresponds to the given alias. It's useful for resolving parameter aliases to their primary names in configuration contexts.

**Parameters**

- **alias** (*str*) -- The parameter alias to look up.
- **ignore_private** (*bool, optional*) -- If True (default), exclude private parameters (those starting with underscore) from the search. If False, include all parameters regardless of privacy.

**Returns**

The canonical parameter name if the alias is found, otherwise None.

**Return type**

str | None

### Example

```
>>> config = Configuration()
>>> config.name_for_alias("channel_priority")
'channel_priority'
>>> config.name_for_alias("unknown_alias")
None
```

**_get_parameter_loader**(*parameter_name*)

Get parameter loader with fallback for missing parameters.

**_reset_cache**()

**register_reset_callaback**(*callback*)

**check_source**(*source*)

**validate_all**()

**static _collect_validation_error**(*func*, **args*, ***kwargs*)

**validate_configuration**()

**post_build_validation**()

**collect_all**()

**describe_parameter**(*parameter_name*)

**list_parameters**(*aliases: bool = False*)

**typify_parameter**(*parameter_name*, *value*, *source*)

**abstract get_descriptions**()

**unique_sequence_map**(*, *unique_key: str*)

Used to validate properties on *Configuration* subclasses defined as a `SequenceParameter(MapParameter())` where the map contains a single key that should be regarded as unique. This decorator will handle removing duplicates and merging to a single sequence.

### constants

Common constants.

**NULL**

**TRACE = 5**

### disk

Common disk utilities.

### io

Common I/O utilities.

## Classes

| | |
|---|---|
| *DeltaSecondsFormatter* | Logging formatter with additional attributes for run time logging. |
| *ContextDecorator* | Base class for a context manager class (implementing __enter__() and __exit__()) that also |
| *SwallowBrokenPipe* | Base class for a context manager class (implementing __enter__() and __exit__()) that also |
| *CaptureTarget* | Constants used for contextmanager captured. |
| *Spinner* | |
| *ProgressBar* | |
| *DummyExecutor* | This is an abstract base class for concrete asynchronous executors. |
| *ThreadLimitedThreadPoolExecutor* | This is an abstract base class for concrete asynchronous executors. |
| *time_recorder* | Base class for a context manager class (implementing __enter__() and __exit__()) that also |

### Functions

| | |
|---|---|
| *dashlist*(iterable[, indent]) | |
| *env_vars*([var_map, callback, stack_callback]) | |
| *env_var*(name, value[, callback, stack_callback]) | |
| *env_unmodified*([callback]) | |
| *captured*([stdout, stderr]) | Capture outputs of sys.stdout and sys.stderr. |
| *argv*(args_list) | |
| *_logger_lock*() | |
| *disable_logger*(logger_name) | |
| *stderr_log_level*(level[, logger_name]) | |
| *attach_stderr_handler*([level, logger_name, propagate, ...]) | Attach a new *stderr* handler to the given logger and configure both. |
| *timeout*(timeout_secs, func, *args[, default_return]) | Enforce a maximum time for a callable to complete. |
| *get_instrumentation_record_file*() | |
| *print_instrumentation_data*() | |

### Attributes

| |
|---|
| *IS_INTERACTIVE* |
| *_FORMATTER* |
| *swallow_broken_pipe* |
| *as_completed* |

#### IS_INTERACTIVE

**class DeltaSecondsFormatter**(*fmt=None*, *datefmt=None*)

Bases: logging.Formatter

Logging formatter with additional attributes for run time logging.

**`delta_secs`**

Elapsed seconds since last log/format call (or creation of logger).

**`relative_created_secs`**

Like *relativeCreated*, time relative to the initialization of the *logging* module but conveniently scaled to seconds as a *float* value.

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument. If datefmt is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of '%', '{' or '$' to specify that you want to use one of %-formatting, `str.format()` ({}) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

**format**(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

**_FORMATTER**

**dashlist**(*iterable*, *indent=2*)

**class ContextDecorator**

Base class for a context manager class (implementing __enter__() and __exit__()) that also makes it a decorator.

**__call__**(*f*)

**class SwallowBrokenPipe**

Bases: *ContextDecorator*

Base class for a context manager class (implementing __enter__() and __exit__()) that also makes it a decorator.

**__enter__**()

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)

**swallow_broken_pipe**

**class CaptureTarget**(**args*, ***kwds*)

Bases: `enum.Enum`

Constants used for contextmanager captured.

Used similarly like the constants PIPE, STDOUT for stdlib's subprocess.Popen.

**STRING**

**STDOUT**

**env_vars**(*var_map=None*, *callback=None*, *stack_callback=None*)

**env_var**(*name*, *value*, *callback=None*, *stack_callback=None*)

**env_unmodified**(*callback=None*)

**captured**(*stdout=CaptureTarget.STRING*, *stderr=CaptureTarget.STRING*)

Capture outputs of sys.stdout and sys.stderr.

If stdout is STRING, capture sys.stdout as a string, if stdout is None, do not capture sys.stdout, leaving it untouched, otherwise redirect sys.stdout to the file-like object given by stdout.

Behave correspondingly for stderr with the exception that if stderr is STDOUT, redirect sys.stderr to stdout target and set stderr attribute of yielded object to None.

```
>>> from conda.common.io import captured
>>> with captured() as c:
...     print("hello world!")
...
>>> c.stdout
'hello world!\n'
```

**Parameters**

- **stdout** -- capture target for sys.stdout, one of STRING, None, or file-like object
- **stderr** -- capture target for sys.stderr, one of STRING, STDOUT, None, or file-like object

**Yields**

*CapturedText* --

**has attributes stdout, stderr which are either strings, None or the** corresponding file-like function argument.

**argv**(*args_list*)

**_logger_lock**()

**disable_logger**(*logger_name*)

**stderr_log_level**(*level*, *logger_name=None*)

**attach_stderr_handler**(*level=WARN*, *logger_name=None*, *propagate=False*, *formatter=None*, *filters=None*)

Attach a new *stderr* handler to the given logger and configure both.

This function creates a new StreamHandler that writes to *stderr* and attaches it to the logger given by *logger_name* (which maybe *None*, in which case the root logger is used). If the logger already has a handler by the name of *stderr*, it is removed first.

The given *level* is set **for the handler**, not for the logger; however, this function also sets the level of the given logger to the minimum of its current effective level and the new handler level, ensuring that the handler will receive the required log records, while minimizing the number of unnecessary log events. It also sets the loggers *propagate* property according to the *propagate* argument. The *formatter* argument can be used to set the formatter of the handler.

**timeout**(*timeout_secs*, *func*, **args*, *default_return=None*, ***kwargs*)

Enforce a maximum time for a callable to complete. Not yet implemented on Windows.

**class Spinner**(*message*, *enabled=True*, *json=False*, *fail_message='failed\n'*)

**Parameters**

- **message** (*str*) -- A message to prefix the spinner with. The string ': ' is automatically appended.
- **enabled** (*bool*) -- If False, usage is a no-op.

- **json** (*bool*) -- If True, will not output non-json to stdout.

**spinner_cycle**

**start**()

**stop**()

**_start_spinning**()

**__enter__**()

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)

**class ProgressBar**(*description*, *enabled=True*, *json=False*, *position=None*, *leave=True*)

**Parameters**

- **description** (*str*) -- The name of the progress bar, shown on left side of output.
- **enabled** (*bool*) -- If False, usage is a no-op.
- **json** (*bool*) -- If true, outputs json progress to stdout rather than a progress bar. Currently, the json format assumes this is only used for "fetch", which maintains backward compatibility with conda 4.3 and earlier behavior.

**classmethod get_lock**()

**update_to**(*fraction*)

**finish**()

**refresh**()

Force refresh i.e. once 100% has been reached

**close**()

**static _tqdm**(**args*, ***kwargs*)

Deferred import so it doesn't hit the *conda activate* paths.

**class DummyExecutor**

Bases: concurrent.futures.Executor

This is an abstract base class for concrete asynchronous executors.

**submit**(*fn*, **args*, ***kwargs*)

Submits a callable to be executed with the given arguments.

Schedules the callable to be executed as fn(*args, **kwargs) and returns a Future instance representing the execution of the callable.

**Returns**

A Future representing the given call.

**map**(*func*, **iterables*)

Returns an iterator equivalent to map(fn, iter).

**Parameters**

- **fn** -- A callable that will take as many arguments as there are passed iterables.
- **timeout** -- The maximum number of seconds to wait. If None, then there is no limit on the wait time.

- **chunksize** -- The size of the chunks the iterable will be broken into before being passed to a child process. This argument is only used by ProcessPoolExecutor; it is ignored by ThreadPoolExecutor.

**Returns**
map(func, *iterables) but the calls may be evaluated out-of-order.

**Return type**
An iterator equivalent to

**Raises**

- **TimeoutError** -- If the entire result iterator could not be generated before the given timeout.
- **Exception** -- If fn(*args) raises for any values.

**shutdown**(*wait=True*)

Clean-up the resources associated with the Executor.

It is safe to call this method several times. Otherwise, no other methods can be called after this one.

**Parameters**

- **wait** -- If True then shutdown will not return until all running futures have finished executing and the resources used by the executor have been reclaimed.
- **cancel_futures** -- If True then shutdown will cancel all pending futures. Futures that are completed or running will not be cancelled.

**class ThreadLimitedThreadPoolExecutor**(*max_workers=10*)

Bases: concurrent.futures.ThreadPoolExecutor

This is an abstract base class for concrete asynchronous executors.

Initializes a new ThreadPoolExecutor instance.

**Parameters**

- **max_workers** -- The maximum number of threads that can be used to execute the given calls.
- **thread_name_prefix** -- An optional name prefix to give our threads.
- **initializer** -- A callable used to initialize worker threads.
- **initargs** -- A tuple of arguments to pass to the initializer.

**submit**(*fn*, **args*, ***kwargs*)

This is an exact reimplementation of the *submit()* method on the parent class, except with an added *try/except* around *self._adjust_thread_count()*. So long as there is at least one living thread, this thread pool will not throw an exception if threads cannot be expanded to *max_workers*.

In the implementation, we use "protected" attributes from concurrent.futures (*_base* and *_WorkItem*). Consider vendoring the whole concurrent.futures library as an alternative to these protected imports.

https://github.com/agronholm/pythonfutures/blob/3.2.0/concurrent/futures/thread.py#L121-L131 # NOQA https://github.com/python/cpython/blob/v3.6.4/Lib/concurrent/futures/thread.py#L114-L124

**as_completed**

**get_instrumentation_record_file**()

class time_recorder(*entry_name=None*, *module_name=None*)

Bases: *ContextDecorator*

Base class for a context manager class (implementing __enter__() and __exit__()) that also makes it a decorator.

record_file

start_time

total_call_num

total_run_time

_set_entry_name(*f*)

__call__(*f*)

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

classmethod log_totals()

_ensure_dir()

print_instrumentation_data()

### iterators

Replacements for parts of the toolz library.

## Functions

| | |
|---|---|
| *groupby_to_dict*(keyfunc, sequence) | A *toolz*-style groupby implementation. |
| *unique*(→ collections.abc.Generator[Any, None, None]) | A *toolz* inspired *unique* implementation. |

groupby_to_dict(*keyfunc*, *sequence*)

A *toolz*-style groupby implementation.

Returns a dictionary of { key: [group] } instead of iterators.

unique(*sequence: collections.abc.Sequence[Any]*) → collections.abc.Generator[Any, None, None]

A *toolz* inspired *unique* implementation.

Returns a generator of unique elements in the sequence

### logic

The basic idea to nest logical expressions is instead of trying to denest things via distribution, we add new variables. So if we have some logical expression expr, we replace it with x and add expr <-> x to the clauses, where x is a new variable, and expr <-> x is recursively evaluated in the same way, so that the final clauses are ORs of atoms.

To use this, create a new Clauses object with the max var, for instance, if you already have [[1, 2, -3]], you would use C = Clause(3). All functions return a new literal, which represents that function, or True or False if the expression can be resolved fully. They may also add new clauses to C.clauses, which will then be delivered to the SAT solver.

All functions take atoms as arguments (an atom is an integer, representing a literal or a negated literal, or boolean constants True or False; that is, it is the callers' responsibility to do the conversion of expressions recursively. This is done because we do not have data structures representing the various logical classes, only atoms.

The polarity argument can be set to True or False if you know that the literal being used will only be used in the positive or the negative, respectively (e.g., you will only use x, not -x). This will generate fewer clauses. It is probably best if you do not take advantage of this directly, but rather through the Require and Prevent functions.

#### Classes

| | |
|---|---|
| *Clauses* | |

#### Functions

| | |
|---|---|
| *minimal_unsatisfiable_subset*(clauses, sat, explicit_specs) | Given a set of clauses, find a minimal unsatisfiable subset (an |

#### Attributes

| | |
|---|---|
| *TRUE* | |
| *FALSE* | |
| *PycoSatSolver* | |
| *PyCryptoSatSolver* | |
| *PySatSolver* | |

**TRUE**

**FALSE**

**PycoSatSolver = 'pycosat'**

PyCryptoSatSolver = 'pycryptosat'

PySatSolver = 'pysat'

class Clauses(*m=0*, *sat_solver=PycoSatSolver*)

property m

property unsat

get_clause_count()

as_list()

_check_variable(*variable*)

_check_literal(*literal*)

add_clause(*clause*)

add_clauses(*clauses*)

name_var(*m*, *name*)

new_var(*name=None*)

from_name(*name*)

from_index(*m*)

_assign(*vals*, *name=None*)

_convert(*x*)

_eval(*func*, *args*, *no_literal_args*, *polarity*, *name*)

Prevent(*what*, **args*)

Require(*what*, **args*)

Not(*x*, *polarity=None*, *name=None*)

And(*f*, *g*, *polarity=None*, *name=None*)

Or(*f*, *g*, *polarity=None*, *name=None*)

Xor(*f*, *g*, *polarity=None*, *name=None*)

ITE(*c*, *t*, *f*, *polarity=None*, *name=None*)

If c Then t Else f.

In this function, if any of c, t, or f are True and False the resulting expression is resolved.

All(*iter*, *polarity=None*, *name=None*)

Any(*vals*, *polarity=None*, *name=None*)

AtMostOne_NSQ(*vals*, *polarity=None*, *name=None*)

AtMostOne_BDD(*vals*, *polarity=None*, *name=None*)

**AtMostOne**(*vals*, *polarity=None*, *name=None*)

**ExactlyOne_NSQ**(*vals*, *polarity=None*, *name=None*)

**ExactlyOne_BDD**(*vals*, *polarity=None*, *name=None*)

**ExactlyOne**(*vals*, *polarity=None*, *name=None*)

**LinearBound**(*equation*, *lo*, *hi*, *preprocess=True*, *polarity=None*, *name=None*)

**sat**(*additional=None*, *includeIf=False*, *names=False*, *limit=0*)

Calculate a SAT solution for the current clause set.

Returned is the list of those solutions. When the clauses are unsatisfiable, an empty list is returned.

**itersolve**(*constraints=None*, *m=None*)

**minimize**(*objective*, *bestsol=None*, *trymax=False*)

**minimal_unsatisfiable_subset**(*clauses*, *sat*, *explicit_specs*)

Given a set of clauses, find a minimal unsatisfiable subset (an unsatisfiable core)

A set is a minimal unsatisfiable subset if no proper subset is unsatisfiable. A set of clauses may have many minimal unsatisfiable subsets of different sizes.

sat should be a function that takes a tuple of clauses and returns True if the clauses are satisfiable and False if they are not. The algorithm will work with any order-reversing function (reversing the order of subset and the order False < True), that is, any function where (A <= B) iff (sat(B) <= sat(A)), where A <= B means A is a subset of B and False < True).

### path

Common path utilities.

### _cygpath

### Functions

| | |
|---|---|
| *nt_to_posix*(→ str) | A fallback implementation of *cygpath --unix*. |
| *_get_root*(→ str) | |
| *_get_RE_WIN_ROOT*(→ re.Pattern) | |
| *_to_unix_root*(→ str) | |
| *_to_unix_mount*(→ str) | |
| *_to_unix_drive*(→ str) | |
| *translate_unix*(→ str) | |
| *posix_to_nt*(→ str) | A fallback implementation of *cygpath --windows*. |
| *_to_win_drive*(→ str) | |
| *_to_win_mount*(→ str) | |
| *_to_win_root*(→ str) | |
| *_resolve_path*(→ str) | |
| *resolve_paths*(→ str) | |

### Attributes

| |
|---|
| *RE_WIN_MOUNT* |
| *RE_WIN_DRIVE* |
| *RE_UNIX_DRIVE* |
| *RE_UNIX_MOUNT* |
| *RE_UNIX_ROOT* |

**nt_to_posix**(*path: conda.common.path.PathType*, *prefix: conda.common.path.PathType | None*, *cygdrive: bool = False*) → str

A fallback implementation of *cygpath --unix*.

**Parameters**

- **path** -- The path to convert.

- **prefix** -- The Windows style prefix directory to use for the conversion. If not provided, no checks for root paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

**_get_root**(*prefix: str*) → str

**_get_RE_WIN_ROOT**(*prefix: str*) → re.Pattern

**_to_unix_root**(*match: re.Match*) → str

**RE_WIN_MOUNT**

**_to_unix_mount**(*match: re.Match*) → str

**RE_WIN_DRIVE**

**_to_unix_drive**(*match: re.Match*, *cygdrive: bool*) → str

**translate_unix**(*match: re.Match*) → str

**posix_to_nt**(*path: conda.common.path.PathType*, *prefix: conda.common.path.PathType | None*, *cygdrive: bool = False*) → str

A fallback implementation of *cygpath --windows*.

**Parameters**

- **path** -- The path to convert.
- **prefix** -- The Windows style prefix directory to use for the conversion. If not provided, no checks for root paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *nt_to_posix*.

**RE_UNIX_DRIVE**

**_to_win_drive**(*match: re.Match*) → str

**RE_UNIX_MOUNT**

**_to_win_mount**(*match: re.Match*) → str

**RE_UNIX_ROOT**

**_to_win_root**(*match: re.Match*, *root: str*) → str

**_resolve_path**(*path: str*, *sep: str*) → str

**resolve_paths**(*paths: str*, *pathsep: str*, *sep: str*) → str

### directories

Common directory utilities.

### Functions

| | |
|---|---|
| *tokenized_startswith*(test_iterable, startswith_iterable) | |
| *get_all_directories*(→ list[tuple[str, Ellipsis]]) | |
| *get_leaf_directories*(→ collections.abc.Sequence[str]) | |
| *explode_directories*(→ set[str]) | |

**tokenized_startswith**(*test_iterable*, *startswith_iterable*)

**get_all_directories**(*files: collections.abc.Iterable[str]*) → list[tuple[str, Ellipsis]]

**get_leaf_directories**(*files: collections.abc.Iterable[str]*) → collections.abc.Sequence[str]

**explode_directories**(*child_directories: collections.abc.Iterable[tuple[str, Ellipsis]]*) → set[str]

#### python

Common Python specific path utilities.

### Functions

| | |
|---|---|
| *pyc_path*(py_path, python_major_minor_version) | This must not return backslashes on Windows as that will break |
| *missing_pyc_files*(python_major_minor_version, files) | |
| *parse_entry_point_def*(ep_definition) | |
| *get_python_short_path*([python_version]) | |
| *get_python_site_packages_short_path*(python_ve | |
| *get_major_minor_version*(string[, with_dot]) | |
| *get_python_noarch_target_path*(source_short_path ...) | |

### Attributes

| |
|---|
| *_VERSION_REGEX* |

**pyc_path**(*py_path*, *python_major_minor_version*)

This must not return backslashes on Windows as that will break tests and leads to an eventual need to make url_to_path return backslashes too and that may end up changing files on disc or to the result of comparisons with the contents of them.

**missing_pyc_files**(*python_major_minor_version*, *files*)

**parse_entry_point_def**(*ep_definition*)

**get_python_short_path**(*python_version=None*)

**get_python_site_packages_short_path**(*python_version*)

**_VERSION_REGEX**

**get_major_minor_version**(*string*, *with_dot=True*)

**get_python_noarch_target_path**(*source_short_path*, *target_site_packages_short_path*)

## windows

Common Windows path utilities.

### Functions

| | |
|---|---|
| *win_path_ok*(path) | |
| *win_path_double_escape*(path) | |
| *win_path_backout*(path) | |
| *_path_to*(→ str \| tuple[str, Ellipsis] \| None) | |
| *win_path_to_unix*(→ str \| tuple[str, Ellipsis] \| None) | Convert Windows paths to Unix paths. |
| *unix_path_to_win*(→ str \| tuple[str, Ellipsis] \| None) | Convert Unix paths to Windows paths. |

**win_path_ok**(*path*)

**win_path_double_escape**(*path*)

**win_path_backout**(*path*)

**_path_to**(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool*, *to_unix: bool*) → str | tuple[str, Ellipsis] | None

**win_path_to_unix**(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, Ellipsis] | None

Convert Windows paths to Unix paths.

> **Note**
>
> Produces unexpected results when run on Unix.

**Parameters**

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

**unix_path_to_win**(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, Ellipsis] | None

Convert Unix paths to Windows paths.

> **Note**
>
> Produces unexpected results when run on Unix.

**Parameters**

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *win_path_to_unix*.

### Functions

| | |
|---|---|
| explode_directories(→ set[str]) | |
| get_all_directories(→ list[tuple[str, Ellipsis]]) | |
| get_leaf_directories(→ collections.abc.Sequence[str]) | |
| tokenized_startswith(test_iterable, startswith_iterable) | |
| get_major_minor_version(string[, with_dot]) | |
| get_python_noarch_target_path(source_short_path, ...) | |
| get_python_short_path([python_version]) | |
| get_python_site_packages_short_path(python_ve | |
| missing_pyc_files(python_major_minor_version, files) | |
| parse_entry_point_def(ep_definition) | |
| pyc_path(py_path, python_major_minor_version) | This must not return backslashes on Windows as that will break |
| unix_path_to_win(→ str \| tuple[str, Ellipsis] \| None) | Convert Unix paths to Windows paths. |
| win_path_backout(path) | |
| win_path_double_escape(path) | |
| win_path_ok(path) | |
| win_path_to_unix(→ str \| tuple[str, Ellipsis] \| None) | Convert Windows paths to Unix paths. |

**explode_directories**(*child_directories: collections.abc.Iterable[tuple[str, Ellipsis]]*) → set[str]

**get_all_directories**(*files: collections.abc.Iterable[str]*) → list[tuple[str, Ellipsis]]

**get_leaf_directories**(*files: collections.abc.Iterable[str]*) → collections.abc.Sequence[str]

**tokenized_startswith**(*test_iterable*, *startswith_iterable*)

**get_major_minor_version**(*string*, *with_dot=True*)

**get_python_noarch_target_path**(*source_short_path*, *target_site_packages_short_path*)

**get_python_short_path**(*python_version=None*)

**get_python_site_packages_short_path**(*python_version*)

**missing_pyc_files**(*python_major_minor_version*, *files*)

**parse_entry_point_def**(*ep_definition*)

**pyc_path**(*py_path*, *python_major_minor_version*)

This must not return backslashes on Windows as that will break tests and leads to an eventual need to make url_to_path return backslashes too and that may end up changing files on disc or to the result of comparisons with the contents of them.

**unix_path_to_win**(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, Ellipsis] | None

Convert Unix paths to Windows paths.

> **Note**
>
> Produces unexpected results when run on Unix.

**Parameters**

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Unused. Present to keep the signature consistent with *win_path_to_unix*.

**win_path_backout**(*path*)

**win_path_double_escape**(*path*)

**win_path_ok**(*path*)

**win_path_to_unix**(*paths: conda.common.path.PathType | conda.common.path.PathsType | None*, *prefix: conda.common.path.PathType | None = None*, ***, *cygdrive: bool = False*) → str | tuple[str, Ellipsis] | None

Convert Windows paths to Unix paths.

> **Note**
>
> Produces unexpected results when run on Unix.

**Parameters**

- **paths** -- The path(s) to convert.
- **prefix** -- The (Windows path-style) prefix directory to use for the conversion. If not provided, no checks for prefix paths will be made.
- **cygdrive** -- Whether to use the Cygwin-style drive prefix.

### pkg_formats

### python

Common Python package format utilities.

**deprecated**

### serialize

YAML and JSON serialization and deserialization functions.

### json

JSON serialization utilities for conda.

## Classes

| | |
|---|---|
| *CondaJSONEncoder* | Extensible JSON <https://json.org> encoder for Python data structures. |

## Functions

| |
|---|
| *dump*(*args, **kwargs) |
| *dumps*(*args, **kwargs) |

## Attributes

| |
|---|
| *load* |
| *loads* |
| *JSONDecodeError* |

**class CondaJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `requests.compat.json.JSONEncoder`

Extensible JSON <https://json.org> encoder for Python data structures.

Supports the following objects and types by default:

| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float | number |
| True | true |
| False | false |
| None | null |

To extend this to recognize other objects, subclass and implement a `.default()` method with another method that returns a serializable object for `o` if possible, otherwise it should call the superclass implementation (to raise `TypeError`).

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float, bool or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

**default**(*obj: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return super().default(o)
```

**dump**(**args*, ***kwargs*)

**dumps**(**args*, ***kwargs*)

**load**

**loads**

**JSONDecodeError**

### Classes

| | |
|---|---|
| *CondaJSONEncoder* | Extensible JSON <https://json.org> encoder for Python data structures. |

### Functions

| | |
|---|---|
| *_yaml_round_trip*() | |
| *_yaml_safe*() | |
| *yaml_round_trip_load*(string) | |
| *yaml_safe_load*(string) | |
| *yaml_round_trip_dump*(object[, stream]) | Dump object to string or stream. |
| *yaml_safe_dump*(object[, stream]) | Dump object to string or stream. |
| *json_dump*(object) | |

### Attributes

| |
|---|
| *deprecated* |
| *loads* |

**deprecated**

**class CondaJSONEncoder**(***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `requests.compat.json.JSONEncoder`

Extensible JSON <https://json.org> encoder for Python data structures.

Supports the following objects and types by default:

| Python | JSON |
|---|---|
| dict | object |
| list, tuple | array |
| str | string |
| int, float | number |
| True | true |
| False | false |
| None | null |

To extend this to recognize other objects, subclass and implement a `.default()` method with another method that returns a serializable object for `o` if possible, otherwise it should call the superclass implementation (to raise `TypeError`).

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float, bool or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

**default**(*obj: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return super().default(o)
```

**loads**

**_yaml_round_trip**()

**_yaml_safe**()

**yaml_round_trip_load**(*string*)

**yaml_safe_load**(*string*)

#### Examples

```
>>> yaml_safe_load("key: value")
{'key': 'value'}
```

**yaml_round_trip_dump**(*object*, *stream=None*)

Dump object to string or stream.

**yaml_safe_dump**(*object*, *stream=None*)

Dump object to string or stream.

**json_dump**(*object*)

### signals

Intercept signals and handle them gracefully.

#### Functions

| |
|---|
| *get_signal_name*(signum) |
| *signal_handler*(handler) |

#### Attributes

| |
|---|
| *INTERRUPT_SIGNALS* |

**INTERRUPT_SIGNALS = ('SIGABRT', 'SIGINT', 'SIGTERM', 'SIGQUIT', 'SIGBREAK')**

**get_signal_name**(*signum*)

### Examples

```
>>> from signal import SIGINT
>>> get_signal_name(SIGINT)
'SIGINT'
```

**signal_handler**(*handler*)

#### toposort

Topological sorting implementation.

### Functions

| | |
|---|---|
| *_toposort*(data) | Dependencies are expressed as a dictionary whose keys are items |
| *pop_key*(data) | Pop an item from the graph that has the fewest dependencies in the case of a tie |
| *_safe_toposort*(data) | Dependencies are expressed as a dictionary whose keys are items |
| *toposort*(data[, safe]) | |

**_toposort**(*data*)

Dependencies are expressed as a dictionary whose keys are items and whose values are a set of dependent items. Output is a list of sets in topological order. The first set consists of items with no dependences, each subsequent set consists of items that depend upon items in the preceding sets.

**pop_key**(*data*)

Pop an item from the graph that has the fewest dependencies in the case of a tie The winners will be sorted alphabetically

**_safe_toposort**(*data*)

Dependencies are expressed as a dictionary whose keys are items and whose values are a set of dependent items. Output is a list of sets in topological order. The first set consists of items with no dependencies, each subsequent set consists of items that depend upon items in the preceding sets.

**toposort**(*data*, *safe=True*)

#### url

Common URL utilities.

### Classes

| | |
|---|---|
| *Url* | Object used to represent a Url. The string representation of this object is a url string. |
| *_SplitUrlParts* | |

### Functions

| | |
|---|---|
| *hex_octal_to_int*(→ int) | |
| *percent_decode*(→ str) | |
| *path_to_url*(→ str) | |
| *urlparse*(→ Url) | |
| *url_to_s3_info*(→ tuple[str, str]) | Convert an s3 url to a tuple of bucket and key. |
| *is_url*(→ bool) | |
| *is_ipv4_address*(→ bool) | |
| *is_ipv6_address*(→ bool) | |
| *is_ip_address*(→ bool) | |
| *join*(*args) | |
| *has_scheme*(→ bool) | |
| *strip_scheme*(→ str) | |
| *mask_anaconda_token*(→ str) | |
| *split_anaconda_token*(url) | |
| *split_platform*(→ tuple[str, str]) | |
| *_split_platform_re*(→ re.Pattern[str]) | |
| *has_platform*(→ bool \| None) | |
| *split_scheme_auth_token*(→ tuple[str \| None, ...) | |
| *split_conda_url_easy_parts*(→ _SplitUrlParts) | |
| *get_proxy_username_and_pass*(→ tuple[str, str]) | |
| *add_username_and_password*(→ str) | Inserts *username* and *password* into provided *url* |
| *maybe_add_auth*(→ str) | Add auth if the url doesn't currently have it. |
| *maybe_unquote*(→ str) | |
| *remove_auth*(→ str) | Remove embedded authentication from URL. |

### Attributes

| | |
|---|---|
| *file_scheme* | def url_to_path(url): |
| *url_attrs* | |
| *join_url* | |

**hex_octal_to_int**(*ho: str*) → int

**percent_decode**(*path: str*) → str

**file_scheme = 'file://'**

def url_to_path(url): assert url.startswith(file_scheme), "{} is not a file-scheme URL".format(url) decoded = percent_decode(url[len(file_scheme):]) if decoded.startswith('/') and decoded[2] == ':':

# A Windows path. decoded.replace('/', '')

return decoded

**path_to_url**(*path: str*) → str

**url_attrs = ('scheme', 'path', 'query', 'fragment', 'username', 'password', 'hostname', 'port')**

**class Url**

Bases: `namedtuple('Url', url_attrs)`

Object used to represent a Url. The string representation of this object is a url string.

This object was inspired by the urllib3 implementation as it gives you a way to construct URLs from various parts. The motivation behind this object was making something that is interoperable with built the *urllib.parse.urlparse* function and has more features than the built-in *ParseResult* object.

Initialize self. See help(type(self)) for accurate signature.

**property auth: str | None**

**property netloc: str | None**

**__str__**() → str

Return str(self).

**as_dict**() → dict[str, str | None]

Provide a public interface for namedtuple's _asdict

**replace**(***kwargs*) → Self

Provide a public interface for namedtuple's _replace

**classmethod from_parse_result**(*parse_result: urllib.parse.ParseResult*) → Self

**urlparse**(*url: str*) → *Url*

**url_to_s3_info**(*url: str*) → tuple[str, str]

Convert an s3 url to a tuple of bucket and key.

#### Examples

```
>>> url_to_s3_info("s3://bucket-name.bucket/here/is/the/key")
('bucket-name.bucket', '/here/is/the/key')
```

**is_url**(*url: Any*) → bool

#### Examples

```
>>> is_url(None)
False
>>> is_url("s3://some/bucket")
True
```

**is_ipv4_address**(*string_ip: str*) → bool

#### Examples

```
>>> [is_ipv4_address(ip) for ip in ('8.8.8.8', '192.168.10.10', '255.255.255.255')]
[True, True, True]
>>> [is_ipv4_address(ip) for ip in ('8.8.8', '192.168.10.10.20', '256.255.255.255',
↪'::1')]
[False, False, False, False]
```

**is_ipv6_address**(*string_ip: str*) → bool

#### Examples

>> [is_ipv6_address(ip) for ip in ('::1', '2001:db8:85a3::370:7334', '1234:'*7+'1234')] [True, True, True] >> [is_ipv6_address(ip) for ip in ('192.168.10.10', '1234:'*8+'1234')] [False, False]

**is_ip_address**(*string_ip: str*) → bool

#### Examples

>> is_ip_address('192.168.10.10') True >> is_ip_address('::1') True >> is_ip_address('www.google.com') False

**join**(**args: str*)

**join_url**

**has_scheme**(*value: str*) → bool

**strip_scheme**(*url: str*) → str

### Examples

```
>>> strip_scheme("https://www.conda.io")
'www.conda.io'
>>> strip_scheme("s3://some.bucket/plus/a/path.ext")
'some.bucket/plus/a/path.ext'
```

**mask_anaconda_token**(*url: str*) → str

**split_anaconda_token**(*url: str*)

### Examples

```
>>> split_anaconda_token("https://1.2.3.4/t/tk-123-456/path")
(u'https://1.2.3.4/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/t//path")
(u'https://1.2.3.4/path', u'')
>>> split_anaconda_token("https://some.domain/api/t/tk-123-456/path")
(u'https://some.domain/api/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/conda/t/tk-123-456/path")
(u'https://1.2.3.4/conda/path', u'tk-123-456')
>>> split_anaconda_token("https://1.2.3.4/path")
(u'https://1.2.3.4/path', None)
>>> split_anaconda_token("https://10.2.3.4:8080/conda/t/tk-123-45")
(u'https://10.2.3.4:8080/conda', u'tk-123-45')
```

**split_platform**(*known_subdirs: collections.abc.Iterable[str]*, *url: str*) → tuple[str, str]

### Examples

```
>>> from conda.base.constants import KNOWN_SUBDIRS
>>> split_platform(KNOWN_SUBDIRS, "https://1.2.3.4/t/tk-123/linux-ppc64le/path")
(u'https://1.2.3.4/t/tk-123/path', u'linux-ppc64le')
```

**_split_platform_re**(*known_subdirs: collections.abc.Iterable[str]*) → re.Pattern[str]

**has_platform**(*url: str*, *known_subdirs: collections.abc.Iterable[str]*) → bool | None

**split_scheme_auth_token**(*url: str*) → tuple[str | None, str | None, str | None, str | None]

### Examples

```
>>> split_scheme_auth_token("https://u:p@conda.io/t/x1029384756/more/path")
('conda.io/more/path', 'https', 'u:p', 'x1029384756')
>>> split_scheme_auth_token(None)
(None, None, None, None)
```

**class _SplitUrlParts**

Bases: NamedTuple

**scheme:** str | None

**auth:** str | None

**token:** str | None

**platform:** str | None

**package_filename:** str | None

**hostname:** str | None

**port:** str | None

**path:** str | None

**query:** str | None

**split_conda_url_easy_parts**(*known_subdirs: collections.abc.Iterable[str]*, *url: str*) → *_SplitUrlParts*

**get_proxy_username_and_pass**(*scheme: str*) → tuple[str, str]

**add_username_and_password**(*url: str*, *username: str*, *password: str*) → str

Inserts *username* and *password* into provided *url*

```
>>> add_username_and_password('https://anaconda.org', 'TestUser', 'Password')
'https://TestUser:Password@anaconda.org'
```

**maybe_add_auth**(*url: str*, *auth: str*, *force: bool = False*) → str

Add auth if the url doesn't currently have it.

By default, does not replace auth if it already exists. Setting `force` to `True` overrides this behavior.

### Examples

```
>>> maybe_add_auth("https://www.conda.io", "user:passwd")
'https://user:passwd@www.conda.io'
>>> maybe_add_auth("https://www.conda.io", "")
'https://www.conda.io'
```

**maybe_unquote**(*url: str*) → str

**remove_auth**(*url: str*) → str

Remove embedded authentication from URL.

```
>>> remove_auth("https://user:password@anaconda.com")
'https://anaconda.com'
```

### core

Code in `conda.core` is the core logic. It is strictly forbidden from having side effects. No printing to stdout or stderr, no disk manipulation, no http requests. All side effects should be implemented through `conda.gateways`. Objects defined in `conda.models` should be heavily preferred for `conda.core` function/method arguments and return values.

Conda modules importable from `conda.core` are

- `conda.common`
- `conda.core`
- `conda.models`
- `conda.gateways`

Conda modules strictly off limits for import within `conda.core` are

- `conda.api`
- `conda.cli`
- `conda.client`

### envs_manager

Tools for managing conda environments.

## Functions

| | |
|---|---|
| *get_user_environments_txt_file*(→ str) | Gets the path to the user's environments.txt file. |
| *register_env*(→ None) | Registers an environment by adding it to environments.txt file. |
| *unregister_env*(→ None) | Unregisters an environment by removing its entry from the environments.txt file if certain conditions are met. |
| *list_all_known_prefixes*(→ list[str]) | Lists all known conda environment prefixes. |
| *query_all_prefixes*(...) | Queries all known prefixes for a given specification. |
| *_clean_environments_txt*(→ tuple[str, Ellipsis]) | Cleans the environments.txt file by removing specified locations. |
| *_rewrite_environments_txt*(→ None) | Rewrites the environments.txt file with the specified prefixes. |

**get_user_environments_txt_file**(*userhome: str = '~'*) → str

Gets the path to the user's environments.txt file.

**Parameters**
**userhome** (*str*) -- The home directory of the user.

**Returns**
Path to the environments.txt file.

**Return type**
str

**register_env**(*location: str*) → None

Registers an environment by adding it to environments.txt file.

**Parameters**
**location** (*str*) -- The file path of the environment to register.

**Returns**
None

**unregister_env**(*location: str*) → None

Unregisters an environment by removing its entry from the environments.txt file if certain conditions are met.

The environment is only unregistered if its associated 'conda-meta' directory exists and contains no significant files other than 'history'. If these conditions are met, the environment's path is removed from environments.txt.

**Parameters**
**location** (*str*) -- The file path of the environment to unregister.

**Returns**
None

**list_all_known_prefixes**() → list[str]

Lists all known conda environment prefixes.

**Returns**
A list of all known conda environment prefixes.

**Return type**
List[str]

**query_all_prefixes**(*spec: str*) → collections.abc.Iterator[tuple[str, tuple]]

Queries all known prefixes for a given specification.

**Parameters**
**spec** (*str*) -- The specification to query for.

**Returns**
An iterator of tuples containing the prefix and the query results.

**Return type**
Iterator[Tuple[str, Tuple]]

**_clean_environments_txt**(*environments_txt_file: str*, *remove_location: str | None = None*) → tuple[str, Ellipsis]

Cleans the environments.txt file by removing specified locations.

**Parameters**

- **environments_txt_file** (*str*) -- The file path of environments.txt.
- **remove_location** (*Optional[str]*) -- Optional location to remove from the file.

**Returns**
A tuple of the cleaned lines.

**Return type**
Tuple[str, ...]

**_rewrite_environments_txt**(*environments_txt_file: str*, *prefixes: list[str]*) → None

Rewrites the environments.txt file with the specified prefixes.

**Parameters**

- **environments_txt_file** (*str*) -- The file path of environments.txt.

- **prefixes** (*List[str]*) -- List of prefixes to write into the file.

**Returns**
None

### index

Tools for fetching the current index.

## Classes

| | |
|---|---|
| *Index* | The `Index` provides information about available packages from all relevant sources. |
| *ReducedIndex* | Index that contains a subset of available packages. |

## Functions

| | |
|---|---|
| *check_allowlist*(→ None) | Check if the given channel URLs are allowed by the context's allowlist. |
| *get_index*(, prepend, platform, use_local, use_cache, ...) | Return the index of packages available on the channels |
| *fetch_index*(→ dict) | Fetch the package index from the specified channels. |
| *dist_str_in_index*(→ bool) | Check if a distribution string matches any package in the index. |
| *_supplement_index_with_prefix*(→ None) | Supplement the given index with information from the specified environment prefix. |
| *_supplement_index_with_cache*(→ None) | Supplement the given index with packages from the cache. |
| *_make_virtual_package*(→ conda.models.records.PackageRecord) | Create a virtual package record. |
| *_supplement_index_with_features*(→ None) | Supplement the given index with virtual feature records. |
| *_supplement_index_with_system*(→ None) | Loads and populates virtual package records from conda plugins |
| *get_archspec_name*(→ str \| None) | Determine the architecture specification name for the current environment. |
| *calculate_channel_urls*(, prepend, platform, use_local) | Calculate the full list of channel URLs to use based on the given parameters. |
| *get_reduced_index*(→ dict) | Generate a reduced package index based on the given specifications. |

### Attributes

*LAST_CHANNEL_URLS*

**LAST_CHANNEL_URLS = []**

**check_allowlist**(*channel_urls: list[str]*) → None

Check if the given channel URLs are allowed by the context's allowlist. :param channel_urls: A list of channel URLs to check against the allowlist. :raises ChannelNotAllowed: If any URL is not in the allowlist. :raises ChannelDenied: If any URL is in the denylist.

**class Index**(*channels: collections.abc.Iterable[str | conda.models.channel.Channel] = ()*, *prepend: bool = True*, *platform: str | None = None*, *subdirs: tuple[str, Ellipsis] | None = None*, *use_local: bool = False*, *use_cache: bool | None = None*, *prefix: str | os.PathLike[str] | pathlib.Path | conda.core.prefix_data.PrefixData | None = None*, *repodata_fn: str | None = context.repodata_fns[-1]*, *use_system: bool = False*)

Bases: `collections.UserDict`

The `Index` provides information about available packages from all relevant sources.

There are four types of sources for package information, namely

**Channels**
represent packages available from standard sources identified with a url, mostly online, but can also be on a local filesystem using the `file://` scheme. Programatically, channels are represented by *conda.models.channel.Channel*, their data is fetched using *conda.core.subdir_data.SubdirData*.

For more information see *What is a "channel"?*.

Individual packages from channels are usually represented by *conda.models.records.PackageRecord*.

**Prefix**
represents packages that are already installed. Every *Index* can be associated with exactly one Prefix, which is the location of one of the conda *Environments*. The package information about the installed packages is represented by *conda.core.prefix_data.PrefixData*.

Individual packages from prefixes are usually represented by *conda.models.records.PrefixRecord*.

**Package Cache**
represents packages that are locally unpacked, but may not be installed in the environment associated with this index. These are usually packages that have been installed in any environment of the local conda installation, but may have been removed from all environments by now.

Individual packages from the package are usually represented by *conda.models.records.PackageCacheRecord*.

**Virtual Packages**
represent properties of the system, not actual conda packages in the normal sense. These are, for example, system packages that inform the solver about the operating system in use, or track features that can be used to steer package priority.

Individual virtual packages are represented by special *conda.models.records.PackageRecord*, see *conda.models.records.PackageRecord.virtual_package()* and *conda.models.records.PackageRecord.feature()*.

Initializes a new index with the desired components.

**Parameters**

- **channels** -- channels identified by canonical names or URLS or Channel objects; for more details, see *conda.models.channel.Channel.from_value()*
- **prepend** -- if `True` (default), add configured channel with higher priority than passed channels; if `False`, do *not* add configured channels.
- **platform** -- see `subdirs`.
- **subdirs** -- platform and subdirs determine the selection of subdirs in the channels; if both are `None`, subdirs is taken from the configuration; if both are given, `subdirs` takes precedence and `platform` is ignored; if only `platform` is given, subdirs will be `(platform, "noarch")`; if `subdirs` is given, subdirs will be `subdirs`.
- **use_local** -- if `True`, add the special "local" channel for locally built packages with lowest priority.
- **use_cache** -- if `True`, add packages from the package cache.
- **prefix** -- associate prefix with this index and add its packages.
- **repodata_fn** -- filename of the repodata, default taken from config, almost always "repodata.json".
- **use_system** -- if `True`, add system packages, that is virtual packages defined by plugins, usually used to make intrinsic information about the system, such as cpu architecture or operating system, available to the solver.

**property cache_entries: tuple[*conda.models.records.PackageCacheRecord*, Ellipsis]**

Contents of the package cache if active.

**Returns**
All packages available from the package cache.

**property system_packages: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]**

System packages provided by plugins.

**Returns**
Identity mapping of the available system packages in a `dict`.

**property features: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]**

Active tracking features.

**Returns**
Identity mapping of the local tracking features in a `dict`.

**property data: dict[*conda.models.records.PackageRecord*, *conda.models.records.PackageRecord*]**

The entire index as a dict; avoid if possible.

> **Warning**
>
> This returns the entire contents of the index as a single identity mapping in a `dict`. This may be convenient, but it comes at a cost because all sources must be fully loaded at significant overhead for *PackageRecord* construction for **every** package.
>
> Hence, all uses of *data*, including all iteration over the entire index, is strongly discouraged.

**reload**(*, *prefix: bool = False*, *cache: bool = False*, *features: bool = False*, *system: bool = False*) → None

Reload one or more of the index components.

Can be used to refresh the index with new information, for example after a new package has been installed into the index.

**Parameters**

- **prefix** -- if `True`, reload the prefix data.
- **cache** -- if `True`, reload the package cache.
- **features** -- if `True`, reload the tracking features.
- **system** -- if `True`, reload the system packages.

**__repr__**() → str

Return repr(self).

**get_reduced_index**(*specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]*) → *ReducedIndex*

Create a reduced index with a subset of packages.

Can be used to create a reduced index as a subset from an existing index.

**Parameters**

**specs** -- the specs that span the subset.

**Returns**

a reduced index with the same sources as this index, but limited to `specs` and their dependency graph.

**_supplement_index_dict_with_prefix**() → None

Supplement the index with information from its prefix.

**_supplement_index_dict_with_cache**() → None

**_realize**() → None

**_retrieve_from_channels**(*key:* conda.models.records.PackageRecord) → *conda.models.records.PackageRecord* | None

**_retrieve_all_from_channels**(*key:* conda.models.records.PackageRecord) → list[*conda.models.records.PackageRecord*]

**_update_from_prefix**(*key:* conda.models.records.PackageRecord, *prec:* conda.models.records.PackageRecord | *None*) → *conda.models.records.PackageRecord* | None

**_update_from_cache**(*key:* conda.models.records.PackageRecord, *prec:* conda.models.records.PackageRecord | *None*) → *conda.models.records.PackageRecord* | None

**__getitem__**(*key:* conda.models.records.PackageRecord) → *conda.models.records.PackageRecord*

**__contains__**(*key:* conda.models.records.PackageRecord) → bool

**__copy__**() → Self

**class ReducedIndex**(*specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec], channels: tuple[str, Ellipsis] = (), prepend: bool = True, platform: str | None = None, subdirs: tuple[str, Ellipsis] | None = None, use_local: bool = False, use_cache: bool | None = None, prefix: str | os.PathLike[str] | pathlib.Path | conda.core.prefix_data.PrefixData | None = None, repodata_fn: str | None = context.repodata_fns[-1], use_system: bool = False*)

Bases: *Index*

Index that contains a subset of available packages.

Like *Index*, this makes information about packages from the same four sources available. However, the contents of the reduced index is limited to a subset of packages relevant to a given specification. This works by taking into account all packages that match the given specification together with their dependencies and their dependencies dependencies, etc.

**Note**

See *Index.get_reduced_index()* for convenient construction.

Initialize a new reduced index.

**Parameters**

- **specs** -- the collection of specifications that span the subset of packages.
- **args** (*all other*) -- see *Index*.

**__repr__**() → str

Return repr(self).

**_derive_reduced_index**() → None

**get_index**(*channel_urls: collections.abc.Iterable[str | conda.models.channel.Channel] = (), prepend: bool = True, platform: str | None = None, use_local: bool = False, use_cache: bool = False, unknown: bool | None = None, prefix: str | None = None, repodata_fn: str = context.repodata_fns[-1]*) → *Index*

Return the index of packages available on the channels

If prepend=False, only the channels passed in as arguments are used. If platform=None, then the current platform is used. If prefix is supplied, then the packages installed in that prefix are added.

**Parameters**

- **channel_urls** -- Channels to include in the index.
- **prepend** -- If False, only the channels passed in are used.
- **platform** -- Target platform for the index.
- **use_local** -- Whether to use local channels.
- **use_cache** -- Whether to use cached index information.
- **unknown** -- Include unknown packages.
- **prefix** -- Path to environment prefix to include in the index.
- **repodata_fn** -- Filename of the repodata file.

**Returns**

A dictionary representing the package index.

**fetch_index**(*channel_urls: list[str]*, *use_cache: bool = False*, *index: dict | None = None*, *repodata_fn: str = context.repodata_fns[-1]*) → dict

Fetch the package index from the specified channels.

**Parameters**

- **channel_urls** -- A list of channel URLs to fetch the index from.
- **use_cache** -- Whether to use the cached index data.
- **index** -- An optional pre-existing index to update.
- **repodata_fn** -- The name of the repodata file.

**Returns**
A dictionary representing the fetched or updated package index.

**dist_str_in_index**(*index: dict[Any, Any]*, *dist_str: str*) → bool

Check if a distribution string matches any package in the index.

**Parameters**

- **index** -- The package index.
- **dist_str** -- The distribution string to match against the index.

**Returns**
True if there is a match; False otherwise.

**_supplement_index_with_prefix**(*index: Index | dict[Any, Any]*, *prefix: str | os.PathLike[str] | pathlib.Path | conda.core.prefix_data.PrefixData*) → None

Supplement the given index with information from the specified environment prefix.

**Parameters**

- **index** -- The package index to supplement.
- **prefix** -- The path to the environment prefix.

**_supplement_index_with_cache**(*index: dict[Any, Any]*) → None

Supplement the given index with packages from the cache.

**Parameters**
**index** -- The package index to supplement.

**_make_virtual_package**(*name: str*, *version: str | None = None*, *build_string: str | None = None*) → *conda.models.records.PackageRecord*

Create a virtual package record.

**Parameters**

- **name** -- The name of the virtual package.
- **version** -- The version of the virtual package, defaults to "0".
- **build_string** -- The build string of the virtual package, defaults to "0".

**Returns**
A PackageRecord representing the virtual package.

**_supplement_index_with_features**(*index: dict[conda.models.records.PackageRecord, conda.models.records.PackageRecord]*, *features: list[str] = []*) → None

Supplement the given index with virtual feature records.

**Parameters**

- **index** -- The package index to supplement.
- **features** -- A list of feature names to add to the index.

**_supplement_index_with_system**(*index: dict[*conda.models.records.PackageRecord, conda.models.records.PackageRecord*]*) → None

Loads and populates virtual package records from conda plugins and adds them to the provided index, unless there is a naming conflict.

**Parameters**

**index** -- The package index to supplement.

**get_archspec_name**() → str | None

Determine the architecture specification name for the current environment.

**Returns**

The architecture name if available, otherwise None.

**calculate_channel_urls**(*channel_urls: tuple[str] = ()*, *prepend: bool = True*, *platform: str | None = None*, *use_local: bool = False*) → list[str]

Calculate the full list of channel URLs to use based on the given parameters.

**Parameters**

- **channel_urls** -- Initial list of channel URLs.
- **prepend** -- Whether to prepend default channels to the list.
- **platform** -- The target platform for the channels.
- **use_local** -- Whether to include the local channel.

**Returns**

The calculated list of channel URLs.

**get_reduced_index**(*prefix: str | None*, *channels: list[str]*, *subdirs: list[str]*, *specs: list[*conda.models.match_spec.MatchSpec*]*, *repodata_fn: str*) → dict

Generate a reduced package index based on the given specifications.

This function is useful for optimizing the solver by reducing the amount of data it needs to consider.

**Parameters**

- **prefix** -- Path to an environment prefix to include installed packages.
- **channels** -- A list of channel names to include in the index.
- **subdirs** -- A list of subdirectories to consider for each channel.
- **specs** -- A list of MatchSpec objects to filter the packages.
- **repodata_fn** -- Filename of the repodata file to use.

**Returns**

A dictionary representing the reduced package index.

### initialize

Backend logic for *conda init*.

Sections in this module are

1. top-level functions
2. plan creators
3. plan runners
4. individual operations
5. helper functions

The top-level functions compose and execute full plans.

A plan is created by composing various individual operations. The plan data structure is a list of dicts, where each dict represents an individual operation. The dict contains two keys--*function* and *kwargs*--where function is the name of the individual operation function within this module.

Each individual operation must

a) return a *Result* (i.e. NEEDS_SUDO, MODIFIED, or NO_CHANGE)

b) have no side effects if context.dry_run is True

c) be verbose and descriptive about the changes being made or proposed is context.verbose

The plan runner functions take the plan (list of dicts) as an argument, and then coordinate the execution of each individual operation. The docstring for *run_plan_elevated()* has details on how that strategy is implemented.

#### Classes

| |
|---|
| *Result* |

#### Functions

| |
|---|
| *install*(conda_prefix) |
| *initialize*(conda_prefix, shells, for_user, for_system, ...) |
| *initialize_dev*(shell[, dev_env_prefix, conda_source_root]) |
| *_initialize_dev_bash*(prefix, env_vars, unset_env_vars) |
| *_initialize_dev_cmdexe*(prefix, env_vars, unset_env_vars) |
| *add_condabin_to_path*(conda_prefix, shells, for_user, ...) |
| *make_install_plan*(conda_prefix) |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| *make_initialize_plan*(conda_prefix, shells, for_user, ...) | Creates a plan for initializing conda in shells. |
| *make_condabin_plan*(conda_prefix, shells, for_user, ...) | Creates a plan to add $PREFIX/condabin to $PATH. |
| *run_plan*(plan) | |
| *run_plan_elevated*(plan) | The strategy of this function differs between unix and Windows. Both strategies use a |
| *run_plan_from_stdin*() | |
| *run_plan_from_temp_file*(temp_path) | |
| *print_plan_results*(plan[, stream]) | |
| *make_entry_point*(target_path, conda_prefix, module, func) | |
| *make_entry_point_exe*(target_path, conda_prefix) | |
| *install_anaconda_prompt*(target_path, conda_prefix, reverse) | |
| *_install_file*(target_path, file_content) | |
| *install_conda_sh*(target_path, conda_prefix) | |
| *install_Scripts_activate_bat*(target_path, conda_prefix) | |
| *install_activate_bat*(target_path, conda_prefix) | |
| *install_deactivate_bat*(target_path, conda_prefix) | |
| *install_activate*(target_path, conda_prefix) | |
| *install_deactivate*(target_path, conda_prefix) | |
| *install_condabin_conda_bat*(target_path, conda_prefix) | |
| *install_library_bin_conda_bat*(target_path, conda_prefix) | |
| *install_condabin_conda_activate_bat*(target_patl ...) | |
| *install_condabin_rename_tmp_bat*(target_path, conda_prefix) | |
| *install_condabin_conda_auto_activate_bat*(targ ...) | |
| *install_condabin_hook_bat*(target_path, conda_prefix) | |
| *install_conda_fish*(target_path, conda_prefix) | |
| *install_conda_psm1*(target_path, conda_prefix) | |
| *install_conda_hook_ps1*(target_path, conda_prefix) | |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| *install_conda_xsh*(target_path, conda_prefix) | |
| *install_conda_csh*(target_path, conda_prefix) | |
| *_config_fish_content*(conda_prefix) | |
| *_config_fish_content_to_add_condabin_to_path* | |
| *init_fish_user*(target_path, conda_prefix[, reverse, ...]) | |
| *_config_xonsh_content*(conda_prefix) | |
| *_config_xonsh_content_to_add_condabin_to_pa* | |
| *init_xonsh_user*(target_path, conda_prefix[, reverse, ...]) | |
| *_bashrc_content*(conda_prefix, shell) | |
| *_bashrc_content_to_add_condabin_to_path*(cond ...) | |
| *init_sh_user*(target_path, conda_prefix, shell[, ...]) | |
| *init_sh_system*(target_path, conda_prefix[, reverse, ...]) | |
| *_read_windows_registry*(target_path) | |
| *_write_windows_registry*(target_path, value_value, ...) | |
| *init_cmd_exe_registry*(target_path, conda_prefix[, reverse]) | |
| *add_condabin_to_path_registry*(target_path, conda_prefix) | |
| *init_long_path*(target_path) | |
| *_powershell_profile_content*(conda_prefix) | |
| *_powershell_profile_content_to_add_condabin_* | |
| *init_powershell_user*(target_path, conda_prefix[, ...]) | |
| *remove_conda_in_sp_dir*(target_path) | |
| *make_conda_egg_link*(target_path, conda_source_root) | |
| *modify_easy_install_pth*(target_path, conda_source_root) | |
| *make_dev_egg_info_file*(target_path) | |
| *make_diff*(old, new) | |

continues on next page

Table 2 – continued from previous page

| |
|---|
| *_get_python_info*(prefix) |

### Attributes

| |
|---|
| *CONDA_INITIALIZE_RE_BLOCK* |
| *CONDA_INITIALIZE_PS_RE_BLOCK* |
| *ContentTypeOptions* |
| *temp_path* |

**CONDA_INITIALIZE_RE_BLOCK = '^# >>> conda initialize >>>(?:\\n|\\r\\n)([\\s\\S]*?)# <<< conda initialize <<<(?:\\n|\\r\\n)?'**

**CONDA_INITIALIZE_PS_RE_BLOCK = '^#region conda initialize(?:\\n|\\r\\n)([\\s\\S]*?)#endregion(?:\\n|\\r\\n)?'**

**class Result**

**NEEDS_SUDO = 'needs sudo'**

**MODIFIED = 'modified'**

**NO_CHANGE = 'no change'**

**ContentTypeOptions**

**install**(*conda_prefix*)

**initialize**(*conda_prefix*, *shells*, *for_user*, *for_system*, *anaconda_prompt*, *reverse=False*)

**initialize_dev**(*shell*, *dev_env_prefix=None*, *conda_source_root=None*)

**_initialize_dev_bash**(*prefix*, *env_vars*, *unset_env_vars*)

**_initialize_dev_cmdexe**(*prefix*, *env_vars*, *unset_env_vars*)

**add_condabin_to_path**(*conda_prefix*, *shells*, *for_user*, *for_system*, *reverse=False*)

**make_install_plan**(*conda_prefix*)

**make_initialize_plan**(*conda_prefix*, *shells*, *for_user*, *for_system*, *anaconda_prompt*, *reverse=False*)

Creates a plan for initializing conda in shells.

Bash: On Linux, when opening the terminal, .bashrc is sourced (because it is an interactive shell). On macOS on the other hand, the .bash_profile gets sourced by default when executing it in Terminal.app. Some other programs do the same on macOS so that's why we're initializing conda in .bash_profile. On Windows, there are multiple ways to open bash depending on how it was installed. Git Bash, Cygwin, and MSYS2 all use .bash_profile by default.

PowerShell: There's several places PowerShell can store its path, depending on if it's Windows PowerShell, PowerShell Core on Windows, or PowerShell Core on macOS/Linux. The easiest way to resolve it is to just ask different possible installations of PowerShell where their profiles are.

**make_condabin_plan**(*conda_prefix*, *shells*, *for_user*, *for_system*, *reverse=False*)

Creates a plan to add $PREFIX/condabin to $PATH.

**run_plan**(*plan*)

**run_plan_elevated**(*plan*)

The strategy of this function differs between unix and Windows. Both strategies use a subprocess call, where the subprocess is run with elevated privileges. The executable invoked with the subprocess is *python -m conda.core.initialize*, so see the *if __name__ == "__main__"* at the bottom of this module.

For unix platforms, we convert the plan list to json, and then call this module with *sudo python -m conda.core.initialize* while piping the plan json to stdin. We collect json from stdout for the results of the plan execution with elevated privileges.

For Windows, we create a temporary file that holds the json content of the plan. The subprocess reads the content of the file, modifies the content of the file with updated execution status, and then closes the file. This process then reads the content of that file for the individual operation execution results, and then deletes the file.

**run_plan_from_stdin**()

**run_plan_from_temp_file**(*temp_path*)

**print_plan_results**(*plan*, *stream=None*)

**make_entry_point**(*target_path*, *conda_prefix*, *module*, *func*)

**make_entry_point_exe**(*target_path*, *conda_prefix*)

**install_anaconda_prompt**(*target_path*, *conda_prefix*, *reverse*)

**_install_file**(*target_path*, *file_content*)

**install_conda_sh**(*target_path*, *conda_prefix*)

**install_Scripts_activate_bat**(*target_path*, *conda_prefix*)

**install_activate_bat**(*target_path*, *conda_prefix*)

**install_deactivate_bat**(*target_path*, *conda_prefix*)

**install_activate**(*target_path*, *conda_prefix*)

**install_deactivate**(*target_path*, *conda_prefix*)

**install_condabin_conda_bat**(*target_path*, *conda_prefix*)

**install_library_bin_conda_bat**(*target_path*, *conda_prefix*)

**install_condabin_conda_activate_bat**(*target_path*, *conda_prefix*)

**install_condabin_rename_tmp_bat**(*target_path*, *conda_prefix*)

**install_condabin_conda_auto_activate_bat**(*target_path*, *conda_prefix*)

**install_condabin_hook_bat**(*target_path*, *conda_prefix*)

**install_conda_fish**(*target_path*, *conda_prefix*)

**install_conda_psm1**(*target_path*, *conda_prefix*)

**install_conda_hook_ps1**(*target_path*, *conda_prefix*)

**install_conda_xsh**(*target_path*, *conda_prefix*)

**install_conda_csh**(*target_path*, *conda_prefix*)

**_config_fish_content**(*conda_prefix*)

**_config_fish_content_to_add_condabin_to_path**(*conda_prefix*)

**init_fish_user**(*target_path*, *conda_prefix*, *reverse=False*, *content_type: ContentTypeOptions = 'initialize'*)

**_config_xonsh_content**(*conda_prefix*)

**_config_xonsh_content_to_add_condabin_to_path**(*conda_prefix*)

**init_xonsh_user**(*target_path*, *conda_prefix*, *reverse=False*, *content_type: ContentTypeOptions = 'initialize'*)

**_bashrc_content**(*conda_prefix*, *shell*)

**_bashrc_content_to_add_condabin_to_path**(*conda_prefix*, *shell*)

**init_sh_user**(*target_path*, *conda_prefix*, *shell*, *reverse=False*, *content_type: ContentTypeOptions = 'initialize'*)

**init_sh_system**(*target_path*, *conda_prefix*, *reverse=False*, *content_type: ContentTypeOptions = 'initialize'*)

**_read_windows_registry**(*target_path*)

**_write_windows_registry**(*target_path*, *value_value*, *value_type*)

**init_cmd_exe_registry**(*target_path*, *conda_prefix*, *reverse=False*)

**add_condabin_to_path_registry**(*target_path*, *conda_prefix*, *reverse=False*)

**init_long_path**(*target_path*)

**_powershell_profile_content**(*conda_prefix*)

**_powershell_profile_content_to_add_condabin_to_path**(*conda_prefix*)

**init_powershell_user**(*target_path*, *conda_prefix*, *reverse=False*, *content_type: ContentTypeOptions = 'initialize'*)

**remove_conda_in_sp_dir**(*target_path*)

**make_conda_egg_link**(*target_path*, *conda_source_root*)

**modify_easy_install_pth**(*target_path*, *conda_source_root*)

**make_dev_egg_info_file**(*target_path*)

**make_diff**(*old*, *new*)

**_get_python_info**(*prefix*)

**temp_path**

### link

Package installation implemented as a series of link/unlink transactions.

## Classes

| | |
|---|---|
| *PrefixSetup* | |
| *ActionGroup* | |
| *PrefixActions* | A container for groups of actions carried out during an UnlinkLinkTransaction. |
| *PrefixActionGroup* | |
| *ChangeReport* | |
| *UnlinkLinkTransaction* | |

## Functions

| | |
|---|---|
| *determine_link_type*(extracted_package_dir, target_prefix) | |
| *make_unlink_actions*(transaction_context, ...) | |
| *match_specs_to_dists*(packages_info_to_link, specs) | |
| *run_script*(→ bool) | Call the post-link (or pre-unlink) script, returning True on success, |
| *messages*(prefix) | |

**determine_link_type**(*extracted_package_dir*, *target_prefix*)

**make_unlink_actions**(*transaction_context*, *target_prefix*, *prefix_record*)

**match_specs_to_dists**(*packages_info_to_link*, *specs*)

**class PrefixSetup**

Bases: NamedTuple

**target_prefix:** str

**unlink_precs:** tuple[*conda.models.records.PackageRecord*, Ellipsis]

**link_precs:** tuple[*conda.models.records.PackageRecord*, Ellipsis]

**remove_specs:** tuple[conda.resolve.MatchSpec, Ellipsis]

**update_specs:** tuple[conda.resolve.MatchSpec, Ellipsis]

**neutered_specs:** tuple[conda.resolve.MatchSpec, Ellipsis]

class ActionGroup

Bases: NamedTuple

**type:** str

**pkg_data:** *conda.models.package_info.PackageInfo* | None

**actions:** collections.abc.Iterable[*conda.core.path_actions.Action*]

**target_prefix:** str

class PrefixActions

A container for groups of actions carried out during an UnlinkLinkTransaction.

**Parameters**

- **remove_menu_action_groups** -- Actions which remove menu items
- **unlink_action_groups** -- Actions which unlink files
- **unregister_action_groups** -- Actions which unregister environment locations
- **link_action_groups** -- Actions which link files
- **register_action_groups** -- Actions which register environment locations
- **compile_action_groups** -- Actions which compile pyc files
- **make_menu_action_groups** -- Actions which create menu items
- **entry_point_action_groups** -- Actions which create python entry points
- **prefix_record_groups** -- Actions which create package json files in conda-meta/
- **initial_action_groups** -- User-defined actions which run before all other actions
- **final_action_groups** -- User-defined actions which run after all other actions

**remove_menu_action_groups:** collections.abc.Iterable[*ActionGroup*]

**unlink_action_groups:** collections.abc.Iterable[*ActionGroup*]

**unregister_action_groups:** collections.abc.Iterable[*ActionGroup*]

**link_action_groups:** collections.abc.Iterable[*ActionGroup*]

**register_action_groups:** collections.abc.Iterable[*ActionGroup*]

**compile_action_groups:** collections.abc.Iterable[*ActionGroup*]

**make_menu_action_groups:** collections.abc.Iterable[*ActionGroup*]

**entry_point_action_groups:** collections.abc.Iterable[*ActionGroup*]

**prefix_record_groups:** collections.abc.Iterable[*ActionGroup*]

**initial_action_groups:** collections.abc.Iterable[*ActionGroup*] = ()

**final_action_groups:** collections.abc.Iterable[*ActionGroup*] = ()

__iter__() → collections.abc.Generator[collections.abc.Iterable[*ActionGroup*], None, None]

class PrefixActionGroup

Bases: NamedTuple

remove_menu_action_groups: collections.abc.Iterable[*ActionGroup*]

unlink_action_groups: collections.abc.Iterable[*ActionGroup*]

unregister_action_groups: collections.abc.Iterable[*ActionGroup*]

link_action_groups: collections.abc.Iterable[*ActionGroup*]

register_action_groups: collections.abc.Iterable[*ActionGroup*]

compile_action_groups: collections.abc.Iterable[*ActionGroup*]

make_menu_action_groups: collections.abc.Iterable[*ActionGroup*]

entry_point_action_groups: collections.abc.Iterable[*ActionGroup*]

prefix_record_groups: collections.abc.Iterable[*ActionGroup*]

class ChangeReport

Bases: NamedTuple

prefix: str

specs_to_remove: collections.abc.Iterable[conda.resolve.MatchSpec]

specs_to_add: collections.abc.Iterable[conda.resolve.MatchSpec]

removed_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

new_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

updated_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

downgraded_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

superseded_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

fetch_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

revised_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*]

class UnlinkLinkTransaction(**setups*)

property nothing_to_do

download_and_extract()

prepare()

verify()

_verify_pre_link_message(*all_link_groups*)

execute()

_get_pfe()

**classmethod _prepare**(*transaction_context*, *target_prefix*, *unlink_precs*, *link_precs*, *remove_specs*, *update_specs*, *neutered_specs*)

**static _verify_individual_level**(*prefix_action_group*)

**static _verify_prefix_level**(*target_prefix_AND_prefix_action_group_tuple*)

**static _verify_transaction_level**(*prefix_setups*)

**_verify**(*prefix_setups*, *prefix_action_groups*)

**_execute**(*all_action_groups*)

**static _execute_actions**(*axngroup*)

**static _execute_post_link_actions**(*axngroup*)

**static _reverse_actions**(*axngroup*, *reverse_from_idx=-1*)

**static _get_python_info**(*target_prefix*, *prefix_recs_to_unlink*, *packages_info_to_link*) → tuple[str | None, str | None]

Return the python version and location of the site-packages directory at the end of the transaction

**static _make_link_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*)

**static _make_entry_point_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*, *link_action_groups*)

**static _make_compile_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*, *link_action_groups*)

**_make_legacy_action_groups**()

**print_transaction_summary**()

**_change_report_str**(*change_report*)

**static _calculate_change_report**(*prefix*, *unlink_precs*, *link_precs*, *download_urls*, *specs_to_remove*, *specs_to_add*)

**run_script**(*prefix: str*, *prec*, *action: str = 'post-link'*, *env_prefix: str = None*, *activate: bool = False*) → bool

Call the post-link (or pre-unlink) script, returning True on success, False on failure.

**messages**(*prefix*)

### package_cache_data

Tools for managing the package cache (previously downloaded packages).

### Classes

| | |
|---|---|
| *PackageCacheType* | This metaclass does basic caching of PackageCache instance objects. |
| *PackageCacheData* | |
| *UrlsData* | |
| *ProgressiveFetchExtract* | |

### Functions

| | |
|---|---|
| *do_cache_action*(prec, cache_action, progress_bar[, ...]) | This function gets called from *ProgressiveFetchExtract.execute*. |
| *do_extract_action*(prec, extract_action, progress_bar) | This function gets called after do_cache_action completes. |
| *do_cleanup*(actions) | |
| *do_reverse*(actions) | |
| *done_callback*(future, actions, progress_bar, exceptions) | |

### Attributes

| | |
|---|---|
| *FileNotFoundError* | |
| *THREADSAFE_EXTRACT* | |
| *EXTRACT_THREADS* | |

**FileNotFoundError**

**THREADSAFE_EXTRACT = False**

**EXTRACT_THREADS**

**class PackageCacheType**

Bases: type

This metaclass does basic caching of PackageCache instance objects.

**__call__**(*pkgs_dir: str | os.PathLike | pathlib.Path*)

Call self as a function.

**class PackageCacheData**(*pkgs_dir*)

property _package_cache_records

property is_writable

_cache_: dict[str, PackageCacheData]

insert(*package_cache_record*)

load()

reload()

get(*package_ref*, *default=NULL*)

remove(*package_ref*, *default=NULL*)

query(*package_ref_or_match_spec*)

iter_records()

classmethod query_all(*package_ref_or_match_spec*, *pkgs_dirs=None*)

classmethod first_writable(*pkgs_dirs=None*)

classmethod writable_caches(*pkgs_dirs=None*)

classmethod read_only_caches(*pkgs_dirs=None*)

classmethod all_caches_writable_first(*pkgs_dirs=None*)

classmethod get_all_extracted_entries()

classmethod get_entry_to_link(*package_ref*)

classmethod tarball_file_in_cache(*tarball_path*, *md5sum=None*, *exclude_caches=()*)

classmethod clear()

tarball_file_in_this_cache(*tarball_path*, *md5sum=None*)

_check_writable()

static _clean_tarball_path_and_get_md5sum(*tarball_path*, *md5sum=None*)

_scan_for_dist_no_channel(*dist_str*)

itervalues()

values()

__repr__()

Return repr(self).

_make_single_record(*package_filename*)

static _dedupe_pkgs_dir_contents(*pkgs_dir_contents*)

class UrlsData(*pkgs_dir*)

__contains__(*url*)

__iter__()

add_url(*url*)

get_url(*package_path*)

class ProgressiveFetchExtract(*link_prefs*)

**Parameters**
**link_prefs** (*tuple[PackageRecord]*) -- A sequence of PackageRecord`s to ensure available in a known package cache, typically for a follow-on :class:`UnlinkLinkTransaction. Here, "available" means the package tarball is both downloaded and extracted to a package directory.

property cache_actions

property extract_actions

static make_actions_for_record(*pref_or_spec*)

prepare()

execute()

Run each action in self.paired_actions. Each action in cache_actions runs before its corresponding extract_actions.

static _progress_bar(*prec_or_spec*, *position=None*, *leave=False*, *context_manager=None*) → *conda.plugins.types.ProgressBarBase*

__hash__()

Return hash(self).

__eq__(*other*)

Return self==value.

do_cache_action(*prec*, *cache_action*, *progress_bar*, *download_total=1.0*, *, *cancelled*)

This function gets called from *ProgressiveFetchExtract.execute*.

do_extract_action(*prec*, *extract_action*, *progress_bar*)

This function gets called after do_cache_action completes.

do_cleanup(*actions*)

do_reverse(*actions*)

done_callback(*future: concurrent.futures.Future*, *actions: tuple[*conda.core.path_actions.CacheUrlAction | conda.core.path_actions.ExtractPackageAction*, Ellipsis]*, *progress_bar:* conda.plugins.types.ProgressBarBase, *exceptions: list[Exception]*, *finish: bool = False*)

### path_actions

Atomic actions that make up a package installation or removal transaction.

## Classes

| | |
|---|---|
| *Action* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *PathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *MultiPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *PrefixPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *CreateInPrefixPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *LinkPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *PrefixReplaceLinkAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *MakeMenuAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *CompileMultiPycAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *AggregateCompileMultiPycAction* | Bunch up all of our compile actions, so that they all get carried out at once. |
| *CreatePythonEntryPointAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *CreatePrefixRecordAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *UpdateHistoryAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *RegisterEnvironmentLocationAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *RemoveFromPrefixPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *UnlinkPathAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *RemoveMenuAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *RemoveLinkedPackageRecordAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *UnregisterEnvironmentLocationAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *CacheUrlAction* | Base class for path manipulation actions, including linking, unlinking, and others. |
| *ExtractPackageAction* | Base class for path manipulation actions, including linking, unlinking, and others. |

### Attributes

| |
|---|
| *FileNotFoundError* |
| *_MENU_RE* |
| *REPR_IGNORE_KWARGS* |

**FileNotFoundError**

**_MENU_RE**

**REPR_IGNORE_KWARGS = ('transaction_context', 'package_info', 'hold_path')**

**class Action**(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*)

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**property verified**

**_verified = False**

**abstract verify**() → Exception | None

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**
On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**abstract execute**() → None

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**abstract reverse**() → None

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**abstract cleanup**() → None

Carry out any post-execution tasks.

**__repr__**()

Return repr(self).

**class PathAction**(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*] | None = None*, *link_precs: collections.abc.Iterable[*conda.models.records.PackageRecord*] | None = None*, *remove_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*, *update_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*, *neutered_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] | None = None*)

Bases: *Action*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**abstract property target_full_path**

class MultiPathAction(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*)

Bases: *Action*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**abstract property target_full_paths**

class PrefixPathAction(*transaction_context*, *target_prefix*, *target_short_path*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked

- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**property target_short_paths**

**property target_full_path**

**class CreateInPrefixPathAction**(*transaction_context*, *package_info*, *source_prefix*, *source_short_path*, *target_prefix*, *target_short_path*)

Bases: *PrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**property source_full_path**

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**cleanup**()

Carry out any post-execution tasks.

class LinkPathAction(*transaction_context*, *package_info*, *extracted_package_dir*, *source_short_path*, *target_prefix*, *target_short_path*, *link_type*, *source_path_data*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

classmethod create_file_link_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*)

classmethod create_directory_actions(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *file_link_actions*)

classmethod create_python_entry_point_windows_exe_action(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *entry_point_def*)

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

reverse()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

class PrefixReplaceLinkAction(*transaction_context*, *package_info*, *extracted_package_dir*, *source_short_path*, *target_prefix*, *target_short_path*, *link_type*, *prefix_placeholder*, *file_mode*, *source_path_data*)

Bases: *LinkPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

verify()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**
On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

execute()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

class MakeMenuAction(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**classmethod create_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*)

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**class CompileMultiPycAction**(*transaction_context*, *package_info*, *target_prefix*, *source_short_paths*, *target_short_paths*)

Bases: *MultiPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**property target_full_paths**

**property source_full_paths**

**classmethod create_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *file_link_actions*)

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**
On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**cleanup**()

Carry out any post-execution tasks.

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**class AggregateCompileMultiPycAction**(**individuals*, ***kw*)

Bases: *CompileMultiPycAction*

Bunch up all of our compile actions, so that they all get carried out at once. This avoids clobbering and is faster when we have several individual packages requiring compilation.

**class CreatePythonEntryPointAction**(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*, *module*, *func*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**classmethod create_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*)

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**class CreatePrefixRecordAction**(*transaction_context*, *package_info*, *target_prefix*, *target_short_path*, *requested_link_type*, *requested_spec*, *all_link_path_actions*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**classmethod create_actions**(*transaction_context*, *package_info*, *target_prefix*, *requested_link_type*, *requested_spec*, *all_link_path_actions*)

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**class UpdateHistoryAction**(*transaction_context*, *target_prefix*, *target_short_path*, *remove_specs*, *update_specs*, *neutered_specs*)

Bases: *CreateInPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**classmethod create_actions**(*transaction_context*, *target_prefix*, *remove_specs*, *update_specs*, *neutered_specs*)

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**class RegisterEnvironmentLocationAction**(*transaction_context*, *target_prefix*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked

- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**abstract property target_full_path**

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**class RemoveFromPrefixPathAction**(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *PrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**
On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**class UnlinkPathAction**(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*, *link_type=LinkType.hardlink*)

Bases: *RemoveFromPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**class RemoveMenuAction**(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *RemoveFromPrefixPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`

2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**classmethod create_actions**(*transaction_context*, *linked_package_data*, *target_prefix*)

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**class RemoveLinkedPackageRecordAction**(*transaction_context*, *linked_package_data*, *target_prefix*, *target_short_path*)

Bases: *UnlinkPathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed

- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**class UnregisterEnvironmentLocationAction**(*transaction_context*, *target_prefix*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**abstract property target_full_path**

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**execute**()

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

`reverse`()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

`cleanup`()

Carry out any post-execution tasks.

`class CacheUrlAction`(*url*, *target_pkgs_dir*, *target_package_basename*, *sha256=None*, *size=None*, *md5=None*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

`property target_full_path`

`verify`()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

`execute`(*progress_update_callback=None*)

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

`_execute_local`(*source_path*, *target_package_cache*, *progress_update_callback=None*)

`_execute_channel`(*target_package_cache*, *progress_update_callback=None*)

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**__str__**()

Return str(self).

**class ExtractPackageAction**(*source_full_path*, *target_pkgs_dir*, *target_extracted_dirname*, *record_or_spec*, *sha256*, *size*, *md5*)

Bases: *PathAction*

Base class for path manipulation actions, including linking, unlinking, and others.

Pre and post-transaction plugins should inherit this class to implement their own verification, execution, reversing, and cleanup steps. These methods are guaranteed to be called in the following order:

1. `verify`
2. `execute`
3. `reverse` (only if `execute` raises an exception)
4. `cleanup`

**Parameters**

- **transaction_context** -- Mapping between target prefixes and PrefixActions instances
- **target_prefix** -- Target prefix for the action
- **unlink_precs** -- Package records to be unlinked
- **link_precs** -- Package records to link
- **remove_specs** -- Specs to be removed
- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**property target_full_path**

**verify**()

Carry out any pre-execution verification.

Should set self._verified = True upon success.

**Returns**

On failure, this function should return (not raise!) an exception

object. At the end of the verification run, all errors will be raised as a CondaMultiError.

**execute**(*progress_update_callback=None*)

Execute the action.

Called after `self.verify()`. If this function raises an exception, `self.reverse()` will be called.

**reverse**()

Reverse what was done in execute.

Called only if `self.execute()` raises an exception.

**cleanup**()

Carry out any post-execution tasks.

**__str__**()

Return str(self).

## portability

Tools for cross-OS portability.

### Functions

| | |
|---|---|
| *_subdir_is_win*(→ bool) | Determine if the given *subdir* corresponds to a Windows operating system. |
| *update_prefix*(→ None) | Update the prefix in a file or directory. |
| *replace_prefix*(→ bytes) | Replaces *placeholder* text with the *new_prefix* provided. The *mode* provided can |
| *binary_replace*(→ bytes) | Replaces occurrences of a search string with a replacement string in a given byte string. |
| *has_pyzzer_entry_point*(→ bool) | Check if the given byte string contains a pyzzer entry point. |
| *replace_pyzzer_entry_point_shebang*(→ bytes) | Code adapted from pyzzer. This is meant to deal with entry point exe's created by distlib, |
| *replace_long_shebang*(→ bytes) | Replace long shebang lines in text mode with a shorter one. |
| *generate_shebang_for_entry_point*(→ str) | This function can be used to generate a shebang line for Python entry points. |

### Attributes

| |
|---|
| *SHEBANG_REGEX* |
| *MAX_SHEBANG_LENGTH* |
| *POPULAR_ENCODINGS* |

**SHEBANG_REGEX = b'^(#!(?:[ ]*)(/(?:\\\\  |[^ \\n\\r\\t])*)(.*))$'**

**MAX_SHEBANG_LENGTH**

**POPULAR_ENCODINGS = ('utf-8', 'utf-16-le', 'utf-16-be', 'utf-32-le', 'utf-32-be')**

**exception _PaddingError**

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**_subdir_is_win**(*subdir: str*) → bool

Determine if the given *subdir* corresponds to a Windows operating system.

**Parameters**
**subdir** -- The subdirectory name which may contain an OS identifier.

**Returns**
Returns True if *subdir* indicates a Windows OS; otherwise, False.

**update_prefix**(*path: str*, *new_prefix: str*, *placeholder: str = PREFIX_PLACEHOLDER*, *mode: conda.models.enums.FileMode = FileMode.text*, *subdir: str = context.subdir*) → None

Update the prefix in a file or directory.

**Parameters**

- **path** -- The path to the file or directory.
- **new_prefix** -- The new prefix to replace the old prefix with.
- **placeholder** -- The placeholder to use for the old prefix. Defaults to PREFIX_PLACEHOLDER.
- **mode** -- The file mode. Defaults to FileMode.text.
- **subdir** -- The subdirectory. Defaults to context.subdir.

**replace_prefix**(*mode: conda.models.enums.FileMode*, *data: bytes*, *placeholder: str*, *new_prefix: str*, *subdir: str = 'noarch'*) → bytes

Replaces *placeholder* text with the *new_prefix* provided. The *mode* provided can either be text or binary.

We use the *POPULAR_ENCODINGS* module level constant defined above to make several passes at replacing the placeholder. We do this to account for as many encodings as possible. If this causes any performance problems in the future, it could potentially be removed (i.e. just using the most popular "utf-8" encoding").

More information/discussion available here: https://github.com/conda/conda/pull/9946

**Parameters**

- **mode** -- The mode of operation.
- **original_data** -- The original data to be updated.
- **placeholder** -- The placeholder to be replaced.
- **new_prefix** -- The new prefix to be used.
- **subdir** -- The subdirectory to be used.

**Returns**
The updated data after prefix replacement.

**binary_replace**(*data: bytes*, *search: bytes*, *replacement: bytes*, *encoding: str = 'utf-8'*, *subdir: str = 'noarch'*) → bytes

Replaces occurrences of a search string with a replacement string in a given byte string.

**Parameters**

- **data** -- The byte string in which to perform the replacements.

- **search** -- The string to search for in the byte string.
- **replacement** -- The string to replace occurrences of the search string with.
- **encoding** -- The encoding to use when encoding and decoding strings. Defaults to "utf-8".
- **subdir** -- The subdirectory to search for. Defaults to "noarch".

**Returns**
The byte string with the replacements made.

**Raises**
**_PaddingError** -- If the padding calculation results in a negative value.

This function performs replacements only for pyzzer-type entry points on Windows. For all other cases, it returns the original byte string unchanged.

**has_pyzzer_entry_point**(*data: bytes*) → bool

Check if the given byte string contains a pyzzer entry point.

**Parameters**
**data** -- The byte string to check.

**Returns**
True if the byte string contains a pyzzer entry point, False otherwise.

**replace_pyzzer_entry_point_shebang**(*all_data: bytes*, *placeholder: bytes*, *new_prefix: str*) → bytes

Code adapted from pyzzer. This is meant to deal with entry point exe's created by distlib, which consist of a launcher, then a shebang, then a zip archive of the entry point code to run. We need to change the shebang. https://bitbucket.org/vinay.sajip/pyzzer/src/5d5740cb04308f067d5844a56fbe91e7a27efccc/pyzzer/__init__.py?at=default&fileviewer=file-view-default#__init__.py-112 # NOQA

**Parameters**
- **all_data** -- The byte string to search for the entry point.
- **placeholder** -- The placeholder string to search for in the entry point.
- **new_prefix** -- The new prefix to replace the placeholder string with.

**Returns**
The updated byte string with the replaced shebang.

**replace_long_shebang**(*mode: conda.models.enums.FileMode*, *data: bytes*) → bytes

Replace long shebang lines in text mode with a shorter one.

**Parameters**
- **mode** -- The file mode.
- **data** -- The byte string to search for the entry point.

**Returns**
The updated byte string with the replaced shebang.

**generate_shebang_for_entry_point**(*executable: str*, *with_usr_bin_env: bool = False*) → str

This function can be used to generate a shebang line for Python entry points.

Use cases: - At install/link time, to generate the *noarch: python* entry points. - conda init uses it to create its own entry point during conda-build

**Parameters**
- **executable** -- The path to the Python executable.

- **with_usr_bin_env** -- Whether to use the */usr/bin/env* approach.

**Returns**
The generated shebang line.

### prefix_data

Tools for managing the packages installed within an environment.

#### Classes

| | |
|---|---|
| *PrefixDataType* | Basic caching of PrefixData instance objects. |
| *PrefixData* | The PrefixData class aims to be the representation of the state |

#### Functions

| | |
|---|---|
| *get_conda_anchor_files_and_records*(...) | Return the anchor files for the conda records of python packages. |
| *python_record_for_prefix*(...) | For the given conda prefix, return the PrefixRecord of the Python installed |
| *get_python_version_for_prefix*(→ str \| None) | For the given conda prefix, return the version of the Python installation |
| *delete_prefix_from_linked_data*(→ bool) | Here, path may be a complete prefix or a dist inside a prefix |

#### Attributes

| |
|---|
| *T* |

**T**

**class PrefixDataType**

Bases: type

Basic caching of PrefixData instance objects.

**__call__**(*prefix_path: str | os.PathLike | pathlib.Path*, *interoperability: bool | None = None*) → *PrefixData*

Call self as a function.

**class PrefixData**(*prefix_path: str | os.PathLike[str] | pathlib.Path*, *interoperability: bool | None = None*)

The PrefixData class aims to be the representation of the state of a conda environment on disk. The directory where the environment lives is called prefix.

This class supports different types of tasks:

- Reading and querying *conda-meta/*.json* files as *PrefixRecord* objects

- Reading and writing environment-specific configuration (env vars, state file, nonadmin markers, etc)
- Existence checks and validations of name, path, and magic files / markers
- Exposing non-conda packages installed in prefix as *PrefixRecord*, via the plugin system

**property name:** `str`

Returns the name of the environment, if available.

If the environment doesn't live in one the configured *envs_dirs*, an empty string is returned. The construct *prefix_data.name or prefix_data.prefix_path* can be helpful in those cases.

**property is_writable:** `bool | None | conda.auxlib._Null`

Check whether the configured path is writable. This is assessed by checking whether *conda-meta/history* is writable. It if is, it is assumed that the rest of the directory tree is writable too.

Note: The value is cached in the instance. Use *.assert_writable()* for a non- cached check.

**property _pip_interop_enabled**

**property _prefix_records:** `dict[str, conda.models.records.PrefixRecord] | None`

**property _python_pkg_record:** `conda.models.records.PrefixRecord | None`

Return the prefix record for the package python.

**_cache_:** `dict[tuple[pathlib.Path, bool | None], PrefixData]`

**classmethod from_name**(*name: str*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from an environment name.

The name will be validated with *PrefixData.validate_name()* if it does not exist.

**Parameters**
**name** -- The name of the environment. Must not contain path separators (/, ).

**Raises**
*CondaValueError* -- If *name* contains a path separator.

**classmethod from_context**(*validate: bool = False*) → *PrefixData*

Creates a PrefixData instance from the path specified by *context.target_prefix*.

The path and name will be validated with *PrefixData.validate_path()* and *PrefixData.validate_name()*, respectively, if *validate* is *True*.

**Parameters**
**validate** -- Whether the path and name should be validated. Useful for environments about to be created.

**__eq__**(*other: Any*) → bool

Return self==value.

**exists**() → bool

Check whether the PrefixData path exists and is a directory.

**is_environment**() → bool

Check whether the PrefixData path is a valida conda environment.

This is assessed by checking if *conda-meta/history* marker file exists.

**is_frozen**() → bool

Check whether the environment is marked as frozen, as per CEP 22.

This is assessed by checking if *conda-meta/frozen* marker file exists.

**is_base**() → bool

Check whether the configured path refers to the *base* environment.

**assert_exists**() → None

Check whether the environment path exists.

**Raises**
***EnvironmentLocationNotFound*** -- If the check returns False.

**assert_environment**() → None

Check whether the environment path exists and is a valid conda environment.

**Raises**
***DirectoryNotACondaEnvironmentError*** -- If the check returns False.

**assert_writable**() → None

Check whether the environment path is a valid conda environment and is writable.

**Raises**
***EnvironmentNotWritableError*** -- If the check returns False.

**assert_not_frozen**() → None

Check whether the environment path is a valid conda environment and is not marked as frozen (as per CEP 22).

**Raises**
***EnvironmentIsFrozenError*** -- If the environment is marked as frozen.

**validate_path**(*expand_path: bool = False*) → None

Validate the path of the environment.

It runs the following checks:

- Make sure the path does not contain *:* or *;* (OS-dependent).
- Disallow immediately nested environments (e.g. *$CONDA_ROOT* and *$CONDA_ROOT/my-env*).
- Warn if there are spaces in the path.

**Parameters**
**expand_path** -- Whether to process ~ and environment variables in the string. The expanded value will replace *.prefix_path*.

**Raises**
***CondaValueError*** -- If the environment contains *:*, *;*, or is nested.

**validate_name**(*allow_base: bool = False*) → None

Validate the name of the environment.

**Parameters**
**allow_base** -- Whether to allow *base* as a valid name.

**Raises**
***CondaValueError*** -- If the name is protected, or if it contains disallowed characters (*/*, ` , `:, *#*).

**load**() → None

**reload**() → *PrefixData*

**_get_json_fn**(*prefix_record:* conda.models.records.PrefixRecord) → str

**insert**(*prefix_record:* conda.models.records.PrefixRecord, *remove_auth: bool = True*) → None

**remove**(*package_name: str*) → None

**get**(*package_name: str*, *default: T = NULL*) → *conda.models.records.PackageRecord* | T

**iter_records**() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**iter_records_sorted**() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**all_subdir_urls**() → set[str]

**query**(*package_ref_or_match_spec:* conda.models.records.PackageRecord | conda.models.match_spec.MatchSpec | *str*) → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**_load_single_record**(*prefix_record_json_path: conda.common.path.PathType*) → None

**_load_site_packages**() → dict[str, *conda.models.records.PrefixRecord*]

**_get_environment_state_file**() → dict[str, dict[str, str]]

**_write_environment_state_file**(*state: dict[str, dict[str, str]]*) → None

**get_environment_env_vars**() → dict[str, str] | dict[bytes, bytes]

**set_environment_env_vars**(*env_vars: dict[str, str]*) → dict[str, str] | None

**unset_environment_env_vars**(*env_vars: dict[str, str]*) → dict[str, str] | None

**set_nonadmin**() → None

Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows).

**get_conda_anchor_files_and_records**(*site_packages_short_path: conda.common.path.PathType*, *python_records: collections.abc.Iterable[*conda.models.records.PrefixRecord*]*) → dict[conda.common.path.PathType, *conda.models.records.PrefixRecord*]

Return the anchor files for the conda records of python packages.

**python_record_for_prefix**(*prefix: os.PathLike*) → *conda.models.records.PrefixRecord* | None

For the given conda prefix, return the PrefixRecord of the Python installed in that prefix.

**get_python_version_for_prefix**(*prefix: os.PathLike*) → str | None

For the given conda prefix, return the version of the Python installation in that prefix.

**delete_prefix_from_linked_data**(*path: str | os.PathLike | pathlib.Path*) → bool

Here, path may be a complete prefix or a dist inside a prefix

### solve

The classic solver implementation.

## Classes

| | |
|---|---|
| *Solver* | A high-level API to conda's solving logic. Three public methods are provided to access a |
| *SolverStateContainer* | |

## Functions

| | |
|---|---|
| *get_pinned_specs*(prefix) | Find pinned specs from file and return a tuple of MatchSpec. |
| *diff_for_unlink_link_precs*([force_reinstall]) | |

**class Solver**(*prefix: str*, *channels: collections.abc.Iterable[*conda.models.channel.Channel*]*, *subdirs: collections.abc.Iterable[str] = ()*, *specs_to_add: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*, *specs_to_remove: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*] = ()*, *repodata_fn: str = REPODATA_FN*, *command=NULL*)

A high-level API to conda's solving logic. Three public methods are provided to access a solution in various forms.

- *solve_final_state()*
- *solve_for_diff()*
- *solve_for_transaction()*

**Parameters**

- **prefix** (*str*) -- The conda prefix / environment location for which the *Solver* is being instantiated.
- **channels** (Sequence[Channel]) -- A prioritized list of channels to use for the solution.
- **subdirs** (*Sequence[str]*) -- A prioritized list of subdirs to use for the solution.
- **specs_to_add** (set[MatchSpec]) -- The set of package specs to add to the prefix.
- **specs_to_remove** (set[MatchSpec]) -- The set of package specs to remove from the prefix.

**_index:** *conda.core.index.ReducedIndex* | None

**_r:** *conda.resolve.Resolve* | None

**solve_for_transaction**(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=NULL*, *should_retry_solve=False*)

Gives an UnlinkLinkTransaction instance that can be used to execute the solution on an environment.

**Parameters**

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) -- See *solve_for_diff()*.
- **should_retry_solve** (*bool*) -- See *solve_final_state()*.

**Return type**
*UnlinkLinkTransaction*

**solve_for_diff**(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *force_reinstall=NULL*, *should_retry_solve=False*) → tuple[tuple[*conda.models.records.PackageRecord*, Ellipsis], tuple[*conda.models.records.PackageRecord*, Ellipsis]]

Gives the package references to remove from an environment, followed by the package references to add to an environment.

**Parameters**

- **deps_modifier** (DepsModifier) -- See *solve_final_state()*.
- **prune** (*bool*) -- See *solve_final_state()*.
- **ignore_pinned** (*bool*) -- See *solve_final_state()*.
- **force_remove** (*bool*) -- See *solve_final_state()*.
- **force_reinstall** (*bool*) --

  **For requested specs_to_add that are already satisfied in the environment,**
  instructs the solver to remove the package and spec from the environment, and then add it back--possibly with the exact package instance modified, depending on the spec exactness.

- **should_retry_solve** (*bool*) -- See *solve_final_state()*.

**Returns**
A two-tuple of PackageRef sequences. The first is the group of packages to remove from the environment, in sorted dependency order from leaves to roots. The second is the group of packages to add to the environment, in sorted dependency order from roots to leaves.

**Return type**
tuple[PackageRef], tuple[PackageRef]

**solve_final_state**(*update_modifier=NULL*, *deps_modifier=NULL*, *prune=NULL*, *ignore_pinned=NULL*, *force_remove=NULL*, *should_retry_solve=False*)

Gives the final, solved state of the environment.

**Parameters**

- **update_modifier** (UpdateModifier) -- An optional flag directing how updates are handled regarding packages already existing in the environment.
- **deps_modifier** (DepsModifier) -- An optional flag indicating special solver handling for dependencies. The default solver behavior is to be as conservative as possible with dependency updates (in the case the dependency already exists in the environment), while

still ensuring all dependencies are satisfied. Options include * NO_DEPS * ONLY_DEPS * UPDATE_DEPS * UPDATE_DEPS_ONLY_DEPS * FREEZE_INSTALLED

- **prune** (*bool*) -- If `True`, the solution will not contain packages that were previously brought into the environment as dependencies but are no longer required as dependencies and are not user-requested.
- **ignore_pinned** (*bool*) -- If `True`, the solution will ignore pinned package configuration for the prefix.
- **force_remove** (*bool*) -- Forces removal of a package without removing packages that depend on it.
- **should_retry_solve** (*bool*) -- Indicates whether this solve will be retried. This allows us to control whether to call find_conflicts (slow) in ssc.r.solve

**Returns**
In sorted dependency order from roots to leaves, the package references for the solved state of the environment.

**Return type**
tuple[PackageRef]

**determine_constricting_specs**(*spec*, *solution_precs*)

**get_request_package_in_solution**(*solution_precs*, *specs_map*)

**get_constrained_packages**(*pre_packages*, *post_packages*, *index_keys*)

**_collect_all_metadata**(*ssc*)

**_remove_specs**(*ssc*)

**_find_inconsistent_packages**(*ssc*)

**_package_has_updates**(*ssc*, *spec*, *installed_pool*)

**_should_freeze**(*ssc*, *target_prec*, *conflict_specs*, *explicit_pool*, *installed_pool*)

**_add_specs**(*ssc*)

**_run_sat**(*ssc*)

**_post_sat_handling**(*ssc*)

**_notify_conda_outdated**(*link_precs*)

**_prepare**(*prepared_specs*) → tuple[*conda.core.index.ReducedIndex*, *conda.resolve.Resolve*]

**class SolverStateContainer**(*prefix*, *update_modifier*, *deps_modifier*, *prune*, *ignore_pinned*, *force_remove*, *should_retry_solve*)

**prefix_data**()

**specs_from_history_map**()

**track_features_specs**()

**pinned_specs**()

**set_repository_metadata**(*index*, *r*)

**_init_solution_precs**()

**working_state_reset**()

**get_pinned_specs**(*prefix*)

Find pinned specs from file and return a tuple of MatchSpec.

**diff_for_unlink_link_precs**(*prefix*, *final_precs*, *specs_to_add=()*, *force_reinstall=NULL*) → tuple[tuple[*conda.models.records.PackageRecord*, Ellipsis], tuple[*conda.models.records.PackageRecord*, Ellipsis]]

## subdir_data

Tools for managing a subdir's repodata.json.

### Classes

| | |
|---|---|
| *SubdirDataType* | |
| *PackageRecordList* | Lazily convert dicts to PackageRecord. |
| *SubdirData* | A class representing the SubdirData. |

### Functions

| | |
|---|---|
| *make_feature_record*(→ conda.models.records.PackageRecord) | Create a feature record based on the given feature name. |

### Attributes

| |
|---|
| *REPODATA_PICKLE_VERSION* |
| *MAX_REPODATA_VERSION* |
| *REPODATA_HEADER_RE* |

**REPODATA_PICKLE_VERSION = 30**

**MAX_REPODATA_VERSION = 2**

**REPODATA_HEADER_RE = b'"(_etag|_mod|_cache_control)":[ ]?"(.*?[^\\\\])"[,}\\s]'**

**class SubdirDataType**

Bases: type

**__call__**(*channel:* conda.models.channel.Channel, *repodata_fn: str = REPODATA_FN*) → *SubdirData*

Call self as a function.

class PackageRecordList(*initlist=None*)

Bases: collections.UserList

Lazily convert dicts to PackageRecord.

This class inherits from the built-in UserList class and provides a way to lazily convert dictionaries to PackageRecord objects. It overrides the __getitem__ method to check if the item at the given index is a PackageRecord object. If not, it converts the dictionary to a PackageRecord object and stores it in the data list.

**Parameters**
**data** -- The list of items

__getitem__(*i: int*) → *conda.models.records.PackageRecord*

Returns the PackageRecord at index i. If i is a slice, returns a new instance of PackageRecordList containing the PackageRecords at the indices in i. If the PackageRecord at index i is not already an instance of PackageRecord, it is converted to one and stored back in the data list.

**Parameters**
**i** -- An integer or slice object indicating the index or indices of the PackageRecord(s) to be returned.

**Returns**
If i is a slice, returns a new instance of PackageRecordList containing the PackageRecords at the indices in i. If i is an integer, returns the PackageRecord at index i.

class SubdirData(*channel:* conda.models.channel.Channel, *repodata_fn: str = REPODATA_FN*, *RepoInterface: type[SubdirData.__init__.RepoInterface] = CondaRepoInterface*)

A class representing the SubdirData.

This class provides functionality for managing and caching SubdirData instances.

**Parameters**

- **channel** -- The channel object
- **repodata_fn** -- The name of the repodata file. Defaults to REPODATA_FN

**Returns**
A SubdirData instance.

Initializes a new instance of the SubdirData class.

**Parameters**

- **channel** -- The channel object.
- **repodata_fn** -- The repodata filename.
- **RepoInterface** -- The RepoInterface class.

property _repo: *conda.gateways.repodata.RepoInterface*

Changes as we mutate self.repodata_fn.

property repo_cache: *conda.gateways.repodata.RepodataCache*

Returns the *RepodataCache* object associated with the current instance of *SubdirData*.

property repo_fetch: *conda.gateways.repodata.RepodataFetch*

Object to get repodata. Not cached since self.repodata_fn is mutable.

Replaces self._repo & self.repo_cache.

**property cache_path_base:** str

Get the base path for caching the repodata.json file.

This method returns the base path for caching the repodata.json file. It is used to construct the full path for caching the file.

**property url_w_repodata_fn:** str

Get the URL with the repodata filename.

This method returns the URL with the repodata filename.

**property cache_path_json:** pathlib.Path

Get the path to the cache file.

This method returns the path to the cache file.

**property cache_path_state:** pathlib.Path

Out-of-band etag and other state needed by the RepoInterface.

Get the path to the cache state file.

This method returns the path to the cache state file.

**property cache_path_pickle:** str

Get the path to the cache pickle file.

This method returns the path to the cache pickle file.

**_cache_:** dict[tuple[str, str], *PackageRecordList* | *SubdirData*]

**classmethod clear_cached_local_channel_data**(*exclude_file: bool = True*) → None

Clear the cached local channel data.

This method is used to clear the cached local channel data. It is primarily used during unit tests to handle changes in the CONDA_USE_ONLY_TAR_BZ2 environment variable during the process lifetime.

**Parameters**

**exclude_file** -- A flag indicating whether to exclude file:// URLs from the cache.

**static query_all**(*package_ref_or_match_spec: conda.core.index.PackageRef | conda.models.match_spec.MatchSpec | str, channels: collections.abc.Iterable[conda.models.channel.Channel | str] | None = None, subdirs: collections.abc.Iterable[str] | None = None, repodata_fn: str = REPODATA_FN*) → tuple[*conda.models.records.PackageRecord*, Ellipsis]

Executes a query against all repodata instances in the channel/subdir matrix.

**Parameters**

- **package_ref_or_match_spec** -- Either an exact *PackageRef* to match against, or a *MatchSpec* query object. A *str* will be turned into a *MatchSpec* automatically.
- **channels** -- An iterable of urls for channels or *Channel* objects. If None, will fall back to *context.channels*.
- **subdirs** -- If None, will fall back to context.subdirs.
- **repodata_fn** -- The filename of the repodata.

**Returns**

A tuple of *PackageRecord* objects.

**query**(*package_ref_or_match_spec: str | conda.models.match_spec.MatchSpec*) → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that queries for a specific package reference or MatchSpec object.

**Parameters**
**package_ref_or_match_spec** -- The package reference or MatchSpec object to query.

**Yields**
PackageRecord objects.

**reload**() → Self

Update the instance with new information.

**load**() → Self

Load the internal state of the SubdirData instance.

This method loads the internal state of the SubdirData instance.

**iter_records**() → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that iterates over package records.

This function checks if the package records are loaded. If not loaded, it loads them. It returns an iterator over the package records. The package_records attribute could potentially be replaced with fully-converted UserList data after going through the entire list.

**_iter_records_by_name**(*name: str*) → collections.abc.Iterator[*conda.models.records.PackageRecord*]

A function that iterates over package records by name.

This function iterates over package records and yields those whose name matches the given name. If include_self is True, it also yields the record with the given name.

**_load**() → dict[str, Any]

Try to load repodata. If e.g. we are downloading *current_repodata.json*, fall back to *repodata.json* when the former is unavailable.

**_pickle_me**() → None

Pickle the object to the specified file.

**_read_local_repodata**(*state: conda.gateways.repodata.RepodataState*) → dict[str, Any]

Read local repodata from the cache and process it.

**_pickle_valid_checks**(*pickled_state: dict[str, Any]*, *mod: str*, *etag: str*) → collections.abc.Iterator[tuple[str, Any, Any]]

Throw away the pickle if these don't all match.

**Parameters**

- **pickled_state** -- The pickled state to compare against.
- **mod** -- The modification information to check.
- **etag** -- The etag to compare against.

**Yields**
Tuples of the form (check_name, pickled_value, current_value).

**_read_pickled**(*state: conda.gateways.repodata.RepodataState*) → dict[str, Any] | None

Read pickled repodata from the cache and process it.

**Parameters**
**state** -- The repodata state.

**Returns**
A dictionary containing the processed pickled repodata, or None if the repodata is not pickled.

**_process_raw_repodata_str**(*raw_repodata_str: str*, *state:* conda.gateways.repodata.RepodataState | *None* = *None*) → dict[str, Any]

State contains information that was previously in-band in raw_repodata_str.

Process the raw repodata string and return the processed repodata.

**Parameters**
**raw_repodata_str** -- The raw repodata string.

**Returns**
A dictionary containing the processed repodata.

**_process_raw_repodata**(*repodata: dict[str, Any]*, *state:* conda.gateways.repodata.RepodataState | *None* = *None*) → dict[str, Any]

Process the raw repodata dictionary and return the processed repodata.

**Parameters**

- **repodata** -- The raw repodata dictionary.
- **state** -- The repodata state. Defaults to None.

**Returns**
A dictionary containing the processed repodata.

**_get_base_url**(*repodata: dict*, *with_credentials: bool* = *True*) → str

In repodata_version=1, .tar.bz2 and .conda artifacts are assumed to be colocated next to repodata.json, in the same server and directory.

In repodata_version=2, repodata.json files can define a 'base_url' field to override that default assumption. See CEP-15 for more details.

This method deals with both cases and returns the appropriate value.

Get the base URL for the given endpoint.

**Parameters**
**endpoint** -- The endpoint for which the base URL is needed.

**Returns**
The base URL corresponding to the provided endpoint.

**make_feature_record**(*feature_name: str*) → *conda.models.records.PackageRecord*

Create a feature record based on the given feature name.

**Parameters**
**feature_name** -- The name of the feature

**Returns**
The created PackageRecord for the feature.

### deprecations

Tools to aid in deprecating code.

## Classes

| | |
|---|---|
| *DeprecationHandler* | Factory to create a deprecation handle for the specified version. |

## Attributes

| |
|---|
| *T* |
| *deprecated* |

**T**

**exception DeprecatedError**

Bases: RuntimeError

Unspecified run-time error.

Initialize self. See help(type(self)) for accurate signature.

**class DeprecationHandler**(*version: str*)

Factory to create a deprecation handle for the specified version.

**Parameters**
**version** -- The version to compare against when checking deprecation statuses.

**_version: str | None**

**_version_tuple: tuple[int, Ellipsis] | None**

**_version_object: packaging.version.Version | None**

**static _get_version_tuple**(*version: str*) → tuple[int, Ellipsis] | None

Return version as non-empty tuple of ints if possible, else None.

**Parameters**
**version** -- Version string to parse.

**_version_less_than**(*version: str*) → bool

Test whether own version is less than the given version.

**Parameters**
**version** -- Version string to compare against.

**__call__**(*deprecate_in: str*, *remove_in: str*, *, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → Callable[[Callable[P, T]], Callable[P, T]]

Deprecation decorator for functions, methods, & classes.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

**argument**(*deprecate_in: str*, *remove_in: str*, *argument: str*, *, *rename: str | None = None*, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → Callable[[Callable[P, T]], Callable[P, T]]

Deprecation decorator for keyword arguments.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **argument** -- The argument to deprecate.
- **rename** -- Optional new argument name.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

**action**(*deprecate_in: str*, *remove_in: str*, *action: ActionType*, *, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = FutureWarning*) → ActionType

Wraps any argparse.Action to issue a deprecation warning.

**module**(*deprecate_in: str*, *remove_in: str*, *, *addendum: str | None = None*, *stack: int = 0*) → None

Deprecation function for modules.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

**constant**(*deprecate_in: str*, *remove_in: str*, *constant: str*, *value: Any*, *, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → None

Deprecation function for module constant/global.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **constant**
- **value**
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

**topic**(*deprecate_in: str*, *remove_in: str*, ***, *topic: str*, *addendum: str | None = None*, *stack: int = 0*, *deprecation_type: type[Warning] = DeprecationWarning*) → None

Deprecation function for a topic.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **topic** -- The topic being deprecated.
- **addendum** -- Optional additional messaging. Useful to indicate what to do instead.
- **stack** -- Optional stacklevel increment.

**_get_module**(*stack: int*) → tuple[types.ModuleType, str]

Detect the module from which we are being called.

**Parameters**
**stack** -- The stacklevel increment.

**Returns**
The module and module name.

**_generate_message**(*deprecate_in: str*, *remove_in: str*, *prefix: str*, *addendum: str | None*, ***, *deprecation_type: type[Warning]*) → tuple[type[Warning] | None, str]

Generate the standardized deprecation message and determine whether the deprecation is pending, active, or past.

**Parameters**

- **deprecate_in** -- Version in which code will be marked as deprecated.
- **remove_in** -- Version in which code is expected to be removed.
- **prefix** -- The message prefix, usually the function name.
- **addendum** -- Additional messaging. Useful to indicate what to do instead.
- **deprecation_type** -- The warning type to use for active deprecations.

**Returns**
The warning category (if applicable) and the message.

**deprecated**

**env**

**env**

Environment object describing the conda environment.yaml file.

### Classes

| | |
|---|---|
| *Dependencies* | A `dict` subclass that parses the raw dependencies into a conda and pip list |
| *EnvironmentYaml* | A class representing an `environment.yaml` file |
| *Environment* | A class representing an `environment.yaml` file |

### Functions

| | |
|---|---|
| *validate_keys*(data, kwargs) | Check for unknown keys, remove them and print a warning |
| *from_environment*(name, prefix[, no_builds, ...]) | Get `EnvironmentYaml` object from prefix |
| *from_yaml*(yamlstr, **kwargs) | Load and return a `EnvironmentYaml` from a given `yaml` string |
| *_expand_channels*(data) | Expands `EnvironmentYaml` variables for the channels found in the `yaml` data |
| *from_file*(filename) | Load and return an `EnvironmentYaml` from a given file |
| *get_filename*(filename) | Expand filename if local path or return the `url` |
| *print_result*(args, prefix, result) | Print the result of an install operation |

### Attributes

| |
|---|
| *VALID_KEYS* |

**VALID_KEYS = ('name', 'dependencies', 'prefix', 'channels', 'variables')**

**validate_keys**(*data*, *kwargs*)

Check for unknown keys, remove them and print a warning

**from_environment**(*name*, *prefix*, *no_builds=False*, *ignore_channels=False*, *from_history=False*)

Get `EnvironmentYaml` object from prefix

**Parameters**

- **name** -- The name of environment
- **prefix** -- The path of prefix
- **no_builds** -- Whether has build requirement
- **ignore_channels** -- whether ignore_channels
- **from_history** -- Whether environment file should be based on explicit specs in history

Returns: EnvironmentYaml object

**from_yaml**(*yamlstr*, ***kwargs*)

Load and return a `EnvironmentYaml` from a given `yaml` string

**_expand_channels**(*data*)

Expands `EnvironmentYaml` variables for the channels found in the `yaml` data

**from_file**(*filename*)

Load and return an `EnvironmentYaml` from a given file

**class Dependencies**(*raw*, **args*, ***kwargs*)

Bases: `dict`

A `dict` subclass that parses the raw dependencies into a conda and pip list

Initialize self. See help(type(self)) for accurate signature.

**parse**()

Parse the raw dependencies into a conda and pip list

**add**(*package_name*)

Add a package to the `EnvironmentYaml`

**class EnvironmentYaml**(*name=None*, *filename=None*, *channels=None*, *dependencies=None*, *prefix=None*, *variables=None*)

A class representing an `environment.yaml` file

**add_channels**(*channels*)

Add channels to the `EnvironmentYaml`

**remove_channels**()

Remove all channels from the `EnvironmentYaml`

**to_dict**(*stream=None*)

Convert information related to the `EnvironmentYaml` into a dictionary

**to_yaml**(*stream=None*)

Convert information related to the `EnvironmentYaml` into a `yaml` string

**save**()

Save the `EnvironmentYaml` data to a `yaml` file

**to_environment_model**() → *conda.models.environment.Environment*

Convert the `Environment` into a `model.Environment` object

**class Environment**(*name=None*, *filename=None*, *channels=None*, *dependencies=None*, *prefix=None*, *variables=None*)

Bases: *EnvironmentYaml*

A class representing an `environment.yaml` file

**get_filename**(*filename*)

Expand filename if local path or return the `url`

**print_result**(*args*, *prefix*, *result*)

Print the result of an install operation

### installers

### base

Dynamic installer loading.

## Functions

| | |
|---|---|
| *get_installer*(name) | Gets the installer for the given environment. |

**get_installer**(*name*)

Gets the installer for the given environment.

Raises: InvalidInstaller if unable to load installer

### conda

Conda-flavored installer.

## Functions

| | |
|---|---|
| *_solve*(→ conda.core.solve.Solver) | Solve the environment. |
| *dry_run*(→ conda.env.env.EnvironmentYaml) | Do a dry run of the environment solve. |
| *install*(→ dict \| None) | Install packages into a conda environment. |

**_solve**(*prefix: str*, *specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, *_*, ***kwargs*) → *conda.core.solve.Solver*

Solve the environment.

**Parameters**

- **prefix** -- Installation target directory
- **specs** -- Package specifications to install
- **args** -- Command-line arguments
- **env** -- Environment object

**Returns**

Solver object

**dry_run**(*specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, *_*, ***kwargs*) → *conda.env.env.EnvironmentYaml*

Do a dry run of the environment solve.

**Parameters**

- **specs** -- Package specifications to install
- **args** -- Command-line arguments

- **env** -- Environment object

**Returns**
Solved environment object

**Return type**
*EnvironmentYaml*

**install**(*prefix: str*, *specs: list[str]*, *args: argparse.Namespace*, *env:* conda.models.environment.Environment, **_*, ***kwargs*) → dict | None

Install packages into a conda environment.

This function handles two main paths: 1. For environments with explicit_specs (from @EXPLICIT files): Bypasses the solver

and directly installs packages using conda.misc.explicit() as required by CEP-23.

2. For regular Environment instances: Uses the solver to determine the optimal package set before installation.

**Parameters**

- **prefix** -- The target installation path for the environment
- **specs** -- Package specifications to install
- **args** -- Command-line arguments from the conda command
- **env** -- Environment object containing dependencies and channels

**Returns**
Installation result information

> **Note**
>
> This implementation follows CEP-23, which states: "When an explicit input file is processed, the conda client SHOULD NOT invoke a solver."

### pip

Pip-flavored installer.

## Functions

| | |
|---|---|
| *_pip_install_via_requirements*(prefix, specs, args, *_, ...) | Installs the pip dependencies in specs using a temporary pip requirements file. |
| *install*(*args, **kwargs) | |

**_pip_install_via_requirements**(*prefix*, *specs*, *args*, **_*, ***kwargs*)

Installs the pip dependencies in specs using a temporary pip requirements file.

**Parameters**

- **prefix** (*string*) -- The path to the python and pip executables.

- **specs** (*iterable of strings*) -- Each element should be a valid pip dependency. See: https://pip.pypa.io/en/stable/user_guide/#requirements-files

  https://pip.pypa.io/en/stable/reference/pip_install/#requirements-file-format

**install**(**args*, ***kwargs*)

### pip_util

Functions related to core conda functionality that relates to pip

NOTE: This modules used to in conda, as conda/pip.py

#### Functions

| | |
|---|---|
| *pip_subprocess*(args, prefix, cwd) | Run pip in a subprocess |
| *get_pip_installed_packages*(stdout) | Return the list of pip packages installed based on the command output |

**pip_subprocess**(*args*, *prefix*, *cwd*)

Run pip in a subprocess

**get_pip_installed_packages**(*stdout*)

Return the list of pip packages installed based on the command output

### specs

### binstar

Define binstar spec.

#### Classes

| | |
|---|---|
| *BinstarSpec* | spec = BinstarSpec('darth/deathstar') |

**class BinstarSpec**(*name=None*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

spec = BinstarSpec('darth/deathstar') spec.can_handle() # => True / False spec.environment # => YAML string spec.env # => conda.models.environment.Environment model spec.msg # => Error messages :raises: EnvironmentFileNotDownloaded

**msg**

**can_handle**() → bool

Validates loader can process environment definition. :return: True or False

**valid_name**() → bool

Validates name :return: True or False

**valid_package**() → bool

Returns True if package has an environment file :return: True or False

**binstar**() → types.ModuleType | None

**file_data**() → list[dict[str, str]]

**environment**() → *conda.env.env.Environment*

**env**() → *conda.models.environment.Environment*

Express the provided environment file as a conda environment object.

**Returns Environment**
the conda environment represented by the file.

**package**()

**username**() → str

**packagename**() → str

### explicit

Define explicit spec.

## Classes

| | |
|---|---|
| *ExplicitSpec* | The ExplicitSpec class handles explicit environment files. These are ones |

**class ExplicitSpec**(*filename: str | None = None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

The ExplicitSpec class handles explicit environment files. These are ones which are marked with the @EXPLICIT marker.

Initialize the explicit specification.

**Parameters**

- **filename** -- Path to the requirements file
- **kwargs** -- Additional arguments

**property env:** *conda.models.environment.Environment*

Build an environment from the explicit file.

**Returns**
An Environment object containing the package specifications

**Raises**
**ValueError** -- If the file cannot be read

can_handle() → bool

Validates that this spec can process the environment definition. This checks if:

- a filename was provided
- the file has the "@EXPLICIT" marker

**Returns**
True if the file can be handled, False otherwise

## requirements

Define requirements.txt spec.

### Classes

| | |
|---|---|
| *RequirementsSpec* | Reads dependencies from requirements files (including explicit files) |

class RequirementsSpec(*filename: str | None = None*, *name: str | None = None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

Reads dependencies from requirements files (including explicit files) and returns an Environment object from it.

Initialize the requirements specification.

**Parameters**
- **filename** -- Path to the requirements file
- **name** -- (Deprecated) Name of the environment
- **kwargs** -- Additional arguments

property name

property environment: *conda.env.env.EnvironmentYaml*

Build an environment from the requirements file.

This method reads the file as a generator and passes it directly to EnvironmentYaml.

**Returns**
An Environment object containing the package specifications

**Raises**
ValueError -- If the file cannot be read

property env: *conda.models.environment.Environment*

Build an environment from the requirements file.

**Returns**
An Environment object containing the package specifications

**Raises**
ValueError -- If the file cannot be read

msg: str | None

**extensions: ClassVar[set[str]]**

**_valid_file**() → bool

Check if the file exists.

**Returns**
True if the file exists, False otherwise

**_valid_name**() → bool

Check if the name is valid.

**Returns**
True if the name is valid, False otherwise

**can_handle**() → bool

Validates that this spec can process the environment definition. This checks if:

- a filename was provided
- the file has a supported extension

**Returns**
True if the file can be handled, False otherwise

### yaml_file

Define YAML spec.

## Classes

| *YamlFileSpec* | **EXPERIMENTAL** |
|---|---|

**class YamlFileSpec**(*filename=None*, ***kwargs*)

Bases: *conda.plugins.types.EnvironmentSpecBase*

**EXPERIMENTAL**

Base class for all env specs.

**property environment:** *conda.env.env.EnvironmentYaml*

**property env:** *conda.models.environment.Environment*

Express the provided environment file as a conda environment object.

**Returns Environment**
the conda environment represented by the file.

**_environment**

**extensions**

**can_handle**()

Validates loader can process environment definition. This can handle if:

- the provided file exists
- the provided file ends in the supported file extensions (.yaml or .yml)

- the env file can be interpreted and transformed into a *conda.env.env.Environment*

**Returns**
True or False

### Functions

| | |
|---|---|
| *get_spec_class_from_file*(→ FileSpecTypes) | Determine spec class to use from the provided `filename` |
| *detect*(→ SpecTypes) | Return the appropriate spec type to use. |

### Attributes

| |
|---|
| *context* |
| *deprecated* |
| *CONDA_SESSION_SCHEMES* |
| *FileSpecTypes* |

**context**

**deprecated**

**exception EnvironmentFileExtensionNotValid**(*filename: os.PathLike*, **args*, ***kwargs*)

Bases: `CondaEnvException`

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileNotFound**(*filename: os.PathLike*, **args*, ***kwargs*)

Bases: `CondaEnvException`

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentSpecPluginNotDetected**(*name: str*, *plugin_names: collections.abc.Iterable[str]*, *autodetect_disabled_plugins: collections.abc.Iterable[str] = ()*, **args*, ***kwargs*)

Bases: *SpecNotFound*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception SpecNotFound**(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**CONDA_SESSION_SCHEMES**

**FileSpecTypes**

**get_spec_class_from_file**(*filename: str*) → FileSpecTypes

Determine spec class to use from the provided `filename`

**Raises**

*EnvironmentFileExtensionNotValid* | *EnvironmentFileNotFound* --

**detect**(*filename: str | None = None*) → SpecTypes

Return the appropriate spec type to use.

**Raises**

*SpecNotFound* -- Raised if no suitable spec class could be found given the input

**exception_handler**

Error handling and error reporting.

### Classes

| |
|---|
| *ExceptionHandler* |

### Functions

| |
|---|
| *conda_exception_handler*(→ T \| int) |

### Attributes

| |
|---|
| *T* |

**T**

**class ExceptionHandler**

**property http_timeout**

**property user_agent**

**property error_upload_url**

**__call__**(*func: Callable[Ellipsis, T]*, **args*, ***kwargs*) → T | int

**write_out**(**content: str*) → None

**handle_exception**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

**handle_application_exception**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

**_print_conda_exception**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → None

**handle_unexpected_exception**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

**handle_reportable_application_exception**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → int

**get_error_report**(*exc_val: BaseException*, *exc_tb: types.TracebackType*) → dict[str, str]

**print_unexpected_error_report**(*error_report: dict[str, str]*) → None

**print_expected_error_report**(*error_report: dict[str, str]*) → None

**_isatty**() → bool

**_upload**(*error_report: dict[str, str]*) → None

Determine whether or not to upload the error report.

**_ask_upload**() → bool

**_execute_upload**(*error_report: dict[str, Any]*) → None

**_post_upload**(*do_upload: bool*) → None

**conda_exception_handler**(*func: Callable[Ellipsis, T]*, **args*, ***kwargs*) → T | int

### exceptions

Conda exceptions.

## Functions

| | |
|---|---|
| *maybe_raise*(error, context) | |
| *print_conda_exception*(exc_val[, exc_tb]) | |
| *_format_exc*([exc_val, exc_tb]) | |

**exception ResolvePackageNotFound**(*bad_deps: collections.abc.Iterable[collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]]*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception LockError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ArgumentError**(*message: str*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**return_code = 2**

**exception Help**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ActivateHelp**

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception DeactivateHelp**

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception GenericHelp**(*command: str*)

Bases: *Help*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaSignalInterrupt**(*signum: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception TooManyArgumentsError**(*expected: int*, *received: int*, *offending_arguments: collections.abc.Iterable[str]*, *optional_message: str = ''*, **args*)

Bases: *ArgumentError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ClobberError**(*message: str*, *path_conflict:* conda.base.constants.PathConflict, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

__repr__()

Return repr(self).

exception BasicClobberError(*source_path: os.PathLike*, *target_path: os.PathLike*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception KnownPackageClobberError(*target_path: os.PathLike*, *colliding_dist_being_linked:* conda.models.records.PackageRecord | *str*, *colliding_linked_dist:* conda.models.records.PackageRecord | *str*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception UnknownPackageClobberError(*target_path: os.PathLike*, *colliding_dist_being_linked:* conda.models.records.PackageRecord | *str*, *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SharedLinkPathClobberError(*target_path: os.PathLike*, *incompatible_package_dists: collections.abc.Iterable*[conda.models.records.PackageRecord | *str*], *context:* conda.base.context.Context)

Bases: *ClobberError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CommandNotFoundError(*command: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PathNotFoundError(*path: os.PathLike*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DirectoryNotFoundError(*path: os.PathLike*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentLocationNotFound**(*location: os.PathLike*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentNameNotFound**(*environment_name: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NoBaseEnvironmentError**

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception DirectoryNotACondaEnvironmentError**(*target_directory: os.PathLike*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaEnvironmentError**(*message: str*, **args*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception DryRunExit**

Bases: *conda.CondaExitZero*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaSystemExit**(**args*)

Bases: *conda.CondaExitZero*, SystemExit

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception PaddingError**(*dist: str*, *placeholder: str*, *placeholder_length: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception LinkError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaOSError**(*message: str*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ProxyError**(*message: str | None = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaIOError**(*message: str*, **args*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaFileIOError**(*filepath: os.PathLike*, *message: str*, **args*)

Bases: *CondaIOError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaKeyError**(*key: Any*, *message: str*, **args*)

Bases: *conda.CondaError*, KeyError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ChannelError**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ChannelNotAllowed**(*channel:* conda.models.channel.Channel)

Bases: *ChannelError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**warning = 'Channel not included in allowlist'**

**exception ChannelDenied**(*channel:* conda.models.channel.Channel)

Bases: *ChannelNotAllowed*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**warning = 'Channel included in denylist'**

**exception UnavailableInvalidChannel**(*channel:* conda.models.channel.Channel | *str*, *status_code:* *str* | *int*, *response:* *requests.models.Response* | *None* = *None*)

Bases: *ChannelError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**status_code:** **str | int**

**exception OperationNotAllowed**(*message:* *str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaImportError**(*message:* *str*)

Bases: *conda.CondaError*, ImportError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ParseError**(*message:* *str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CouldntParseError**(*reason:* *str*)

Bases: *ParseError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ChecksumMismatchError**(*url:* *str*, *target_full_path:* *os.PathLike*, *checksum_type:* *str*, *expected_checksum:* *str*, *actual_checksum:* *str*, *partial_download:* *bool* = *False*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception PackageNotInstalledError**(*prefix:* *os.PathLike*, *package_name:* *str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaHTTPError**(*message:* *str*, *url:* *str*, *status_code:* *int* | *str*, *reason:* *str*, *elapsed_time:* *datetime.timedelta* | *str*, *response:* *requests.Response* | *None* = *None*, *caused_by:* *Any* = *None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception CondaSSLError(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception AuthenticationError(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception PackagesNotFoundError(*packages: collections.abc.Iterable[*conda.models.match_spec.MatchSpec | conda.models.records.PackageRecord | *str]*, *channel_urls: collections.abc.Iterable[str] = ()*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception UnsatisfiableError(*bad_deps: collections.abc.Iterable[collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]]*, *chains: bool = True*, *strict: bool = False*)

Bases: *conda.CondaError*

An exception to report unsatisfiable dependencies.

**Parameters**

- **bad_deps** -- a list of tuples of objects (likely MatchSpecs).
- **chains** -- (optional) if True, the tuples are interpreted as chains of dependencies, from top level to bottom. If False, the tuples are interpreted as simple lists of conflicting specs.

**Returns**

Raises an exception with a formatted message detailing the unsatisfiable specifications.

Initialize self. See help(type(self)) for accurate signature.

_format_chain_str(*bad_deps: collections.abc.Iterable[collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]]*)

exception RemoveError(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception DisallowedPackageError(*package_ref:* conda.models.records.PackageRecord, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception SpecsConfigurationConflictError(*requested_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]*, *pinned_specs: collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]*, *prefix: os.PathLike*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaIndexError**(*message: str*)

Bases: *conda.CondaError*, IndexError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaValueError**(*message: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CyclicalDependencyError**(*packages_with_cycles: collections.abc.Iterable*[conda.models.records.PackageRecord], ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CorruptedEnvironmentError**(*environment_location: os.PathLike*, *corrupted_file: os.PathLike*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaHistoryError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaUpgradeError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaVerificationError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception SafetyError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaMemoryError**(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*, MemoryError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NotWritableError**(*path: os.PathLike*, *errno: int*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NoWritableEnvsDirError**(*envs_dirs: collections.abc.Iterable[os.PathLike]*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NoWritablePkgsDirError**(*pkgs_dirs: collections.abc.Iterable[os.PathLike]*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentIsFrozenError**(*prefix: os.PathLike*, *message: str = ''*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentNotWritableError**(*environment_location: os.PathLike*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaDependencyError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception BinaryPrefixReplacementError**(*path: os.PathLike*, *placeholder: str*, *new_prefix: os.PathLike*, *original_data_length: int*, *new_data_length: int*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception InvalidSpec**(*message: str*, ***kwargs*)

Bases: *conda.CondaError*, ValueError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception InvalidVersionSpec**(*invalid_spec: str | conda.models.match_spec.MatchSpec*, *details: str*)

Bases: *InvalidSpec*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception InvalidMatchSpec**(*invalid_spec: str | conda.models.match_spec.MatchSpec*, *details: str*)

Bases: *InvalidSpec*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EncodingError**(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NoSpaceLeftError**(*caused_by: Any*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaEnvException**(*message: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileNotFound**(*filename: os.PathLike*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileExtensionNotValid**(*filename: os.PathLike*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileTypeMismatchError**(*file_types: dict[str, str]*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileEmpty**(*filename: os.PathLike*, **args*, ***kwargs*)

Bases: *CondaEnvException*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentFileNotDownloaded**(*username: str*, *packagename: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception PluginError**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception SpecNotFound**(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception EnvironmentSpecPluginNotDetected**(*name: str*, *plugin_names: collections.abc.Iterable[str]*, *autodetect_disabled_plugins: collections.abc.Iterable[str] = ()*, **args*, ***kwargs*)

Bases: *SpecNotFound*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception SpecNotFoundInPackageCache**(*msg: str*, **args*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**maybe_raise**(*error: BaseException*, *context:* conda.base.context.Context)

**print_conda_exception**(*exc_val:* conda.CondaError, *exc_tb: types.TracebackType | None = None*)

**_format_exc**(*exc_val: BaseException | None = None*, *exc_tb: types.TracebackType | None = None*)

**exception InvalidInstaller**(*name: str*)

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception OfflineError**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*, RuntimeError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

### exports

Backported exports for conda-build.

## Classes

| | |
|---|---|
| *Completer* | |
| *InstalledPackages* | |

## Functions

| | |
|---|---|
| *__getattr__*(→ Any) | |
| *iteritems*(d, **kw) | |
| *rm_rf*(path[, max_retries, trash]) | |
| *hash_file*(_) | |
| *verify*(_) | |
| *get_index*([channel_urls, prepend, platform, ...]) | |
| *package_cache*() | |
| *symlink_conda*(prefix, root_dir[, shell]) | |
| *_symlink_conda_hlp*(prefix, root_dir, where, symlink_fn) | |
| *win_conda_bat_redirect*(src, dst, shell) | Special function for Windows XP where the *CreateSymbolicLink* |
| *linked_data*(prefix[, ignore_channels]) | Return a dictionary of the linked packages in prefix. |
| *linked*(prefix[, ignore_channels]) | Return the Dists of linked packages in prefix. |
| *is_linked*(prefix, dist) | Return the install metadata for a linked package in a prefix, or None |
| *download*(url, dst_path[, session, md5sum, urlstxt, ...]) | |

### Attributes

| |
|---|
| *non_x86_linux_machines* |
| *get_default_urls* |
| *arch_name* |
| *binstar_upload* |
| *bits* |
| *default_prefix* |
| *default_python* |
| *envs_dirs* |
| *pkgs_dirs* |
| *platform* |
| *root_dir* |
| *root_writable* |
| *subdir* |
| *conda_build* |
| *get_rc_urls* |
| *get_local_urls* |
| *load_condarc* |
| *PaddingError* |
| *LinkError* |
| *CondaOSError* |
| *CondaFileNotFoundError* |
| *PY3* |
| *string_types* |
| *text_type* |

non_x86_linux_machines

get_default_urls

arch_name

binstar_upload

bits

default_prefix

default_python

envs_dirs

pkgs_dirs

platform

root_dir

root_writable

subdir

conda_build

get_rc_urls

get_local_urls

load_condarc

PaddingError

LinkError

CondaOSError

CondaFileNotFoundError

PY3 = True

string_types

text_type

__getattr__(*name: str*) → Any

iteritems(*d*, ***kw*)

class Completer

    get_items()

    __contains__(*item*)

    __iter__()

class InstalledPackages

rm_rf(*path*, *max_retries=5*, *trash=True*)

**hash_file**(_)

**verify**(_)

**get_index**(*channel_urls=()*, *prepend=True*, *platform=None*, *use_local=False*, *use_cache=False*, *unknown=None*, *prefix=None*)

**package_cache**()

**symlink_conda**(*prefix*, *root_dir*, *shell=None*)

**_symlink_conda_hlp**(*prefix*, *root_dir*, *where*, *symlink_fn*)

**win_conda_bat_redirect**(*src*, *dst*, *shell*)

Special function for Windows XP where the *CreateSymbolicLink* function is not available.

Simply creates a *.bat* file at *dst* which calls *src* together with all command line arguments.

Works of course only with callable files, e.g. *.bat* or *.exe* files.

**linked_data**(*prefix*, *ignore_channels=False*)

Return a dictionary of the linked packages in prefix.

**linked**(*prefix*, *ignore_channels=False*)

Return the Dists of linked packages in prefix.

**is_linked**(*prefix*, *dist*)

Return the install metadata for a linked package in a prefix, or None if the package is not linked in the prefix.

**download**(*url*, *dst_path*, *session=None*, *md5sum=None*, *urlstxt=False*, *retries=3*, *sha256=None*, *size=None*)

### gateways

Gateways isolate interaction of conda code with the outside world. Disk manipulation, database interaction, and remote requests should all be through various gateways. Functions and methods in `conda.gateways` must use `conda.models` for arguments and return values.

Conda modules importable from `conda.gateways` are

- `conda.common`
- `conda.models`
- `conda.gateways`

Conda modules off limits for import within `conda.gateways` are

- `conda.api`
- `conda.cli`
- `conda.client`
- `conda.core`

Conda modules strictly prohibited from importing `conda.gateways` are

- `conda.api`
- `conda.cli`
- `conda.client`

### anaconda_client

Anaconda-client (binstar) token management for CondaSession.

#### Functions

| | |
|---|---|
| *replace_first_api_with_conda*(url) | |
| *_get_binstar_token_directory*() | |
| *read_binstar_tokens*() | |
| *set_binstar_token*(url, token) | |
| *remove_binstar_token*(url) | |

**replace_first_api_with_conda**(*url*)

**_get_binstar_token_directory**()

**read_binstar_tokens**()

**set_binstar_token**(*url*, *token*)

**remove_binstar_token**(*url*)

### connection

### adapters

### ftp

Defines FTP transport adapter for CondaSession (requests.Session).

Taken from requests-ftp (https://github.com/Lukasa/requests-ftp/blob/master/requests_ftp/ftp.py).

### Classes

| | |
|---|---|
| *FTPAdapter* | A Requests Transport Adapter that handles FTP urls. |

### Functions

| | |
|---|---|
| *_new_makepasv*(self) | |
| *data_callback_factory*(variable) | Returns a callback suitable for use by the FTP library. This callback |
| *build_text_response*(request, data, code) | Build a response for textual data. |
| *build_binary_response*(request, data, code) | Build a response for data whose encoding is unknown. |
| *build_response*(request, data, code, encoding) | Builds a response object from the data returned by ftplib, using the |
| *get_status_code_from_code_response*(code) | Handle complicated code response, even multi-lines. |

### Attributes

| |
|---|
| *_old_makepasv* |

**_old_makepasv**

**_new_makepasv**(*self*)

**class FTPAdapter**

Bases: `conda.gateways.connection.BaseAdapter`

A Requests Transport Adapter that handles FTP urls.

**send**(*request*, ***kwargs*)

Sends a PreparedRequest object over FTP. Returns a response object.

**close**()

Dispose of any internal state.

**list**(*path*, *request*)

Executes the FTP LIST command on the given path.

**retr**(*path*, *request*)

Executes the FTP RETR command on the given path.

**nlst**(*path*, *request*)

Executes the FTP NLST command on the given path.

**get_username_password_from_header**(*request*)

Given a PreparedRequest object, reverse the process of adding HTTP Basic auth to obtain the username and password. Allows the FTP adapter to piggyback on the basic auth notation without changing the control flow.

**get_host_and_path_from_url**(*request*)

Given a PreparedRequest object, split the URL in such a manner as to determine the host and the path. This is a separate method to wrap some of urlparse's craziness.

**data_callback_factory**(*variable*)

Returns a callback suitable for use by the FTP library. This callback will repeatedly save data into the variable provided to this function. This variable should be a file-like structure.

**build_text_response**(*request*, *data*, *code*)

Build a response for textual data.

**build_binary_response**(*request*, *data*, *code*)

Build a response for data whose encoding is unknown.

**build_response**(*request*, *data*, *code*, *encoding*)

Builds a response object from the data returned by ftplib, using the specified encoding.

**get_status_code_from_code_response**(*code*)

Handle complicated code response, even multi-lines.

We get the status code in two ways: - extracting the code from the last valid line in the response - getting it from the 3 first digits in the code After a comparison between the two values, we can safely set the code or raise a warning. .. rubric:: Examples

- get_status_code_from_code_response('200 Welcome') == 200
- multi_line_code = '226-File successfully transferredn226 0.000 seconds' get_status_code_from_code_response(multi_line_code) == 226
- multi_line_with_code_conflicts = '200-File successfully transferredn226 0.000 seconds' get_status_code_from_code_response(multi_line_with_code_conflicts) == 226

**For more detail see RFC 959, page 36, on multi-line responses:**
https://www.ietf.org/rfc/rfc959.txt "Thus the format for multi-line replies is that the first line

> will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code."

### http

Defines HTTP transport adapter for CondaSession (requests.Session).

Closely derived from pip:

https://github.com/pypa/pip/blob/8c24fd2a80bad21aa29aec02fb48bd89a1e8f5e1/src/pip/_internal/network/session.py#L254

Under the MIT license:

### Classes

| | |
|---|---|
| *_SSLContextAdapterMixin* | Mixin to add the `ssl_context` constructor argument to HTTP adapters. |
| *HTTPAdapter* | Mixin to add the `ssl_context` constructor argument to HTTP adapters. |

**class _SSLContextAdapterMixin**(*, *ssl_context: ssl.SSLContext | None = None*, ***kwargs: Any*)

Mixin to add the `ssl_context` constructor argument to HTTP adapters.

The additional argument is forwarded directly to the pool manager. This allows us to dynamically decide what SSL store to use at runtime, which is used to implement the optional `truststore` backend.

**init_poolmanager**(*connections: int*, *maxsize: int*, *block: bool = DEFAULT_POOLBLOCK*, ***pool_kwargs: Any*) → urllib3.PoolManager

**class HTTPAdapter**(*, *ssl_context: ssl.SSLContext | None = None*, ***kwargs: Any*)

Bases: *_SSLContextAdapterMixin*, `conda.gateways.connection.HTTPAdapter`

Mixin to add the `ssl_context` constructor argument to HTTP adapters.

The additional argument is forwarded directly to the pool manager. This allows us to dynamically decide what SSL store to use at runtime, which is used to implement the optional `truststore` backend.

## localfs

Defines local filesystem transport adapter for CondaSession (requests.Session).

### Classes

| | |
|---|---|
| *LocalFSAdapter* | The Base Transport Adapter |

**class LocalFSAdapter**

Bases: `conda.gateways.connection.BaseAdapter`

The Base Transport Adapter

**send**(*request*, *stream=None*, *timeout=None*, *verify=None*, *cert=None*, *proxies=None*)

Sends PreparedRequest object. Returns Response object.

**Parameters**

- **request** -- The `PreparedRequest` being sent.
- **stream** -- (optional) Whether to stream the request content.

- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

**close**()

Cleans up adapter specific items.

### s3

Defines S3 transport adapter for CondaSession (requests.Session).

#### Classes

| | |
|---|---|
| *S3Adapter* | The Base Transport Adapter |

#### Attributes

| |
|---|
| *stderrlog* |

**stderrlog**

**class S3Adapter**

Bases: conda.gateways.connection.BaseAdapter

The Base Transport Adapter

**send**(*request: conda.gateways.connection.PreparedRequest*, *stream: bool = False*, *timeout: None | float | tuple[float, float] | tuple[float, None] = None*, *verify: bool | str = True*, *cert: None | bytes | str | tuple[bytes | str, bytes | str] = None*, *proxies: dict[str, str] | None = None*) → conda.gateways.connection.Response

Sends PreparedRequest object. Returns Response object.

**Parameters**

- **request** -- The PreparedRequest being sent.
- **stream** -- (optional) Whether to stream the request content.
- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

**close**()

Cleans up adapter specific items.

**_send_boto3**(*resp: conda.gateways.connection.Response*, *request: conda.gateways.connection.PreparedRequest*) → conda.gateways.connection.Response

**_write_tempfile**(*writer_callable*)

### download

Download logic for conda indices and packages.

## Classes

| | |
|---|---|
| *TmpDownload* | Context manager to handle downloads to a tempfile. |

## Functions

| | |
|---|---|
| *disable_ssl_verify_warning*() | |
| *download*(url, target_full_path[, md5, sha256, size, ...]) | |
| *download_inner*(url, target_full_path, md5, sha256, ...) | |
| *download_partial_file*(target_full_path, *, url, ...) | Create or open locked partial download file, moving onto target_full_path |
| *download_http_errors*(url) | Exception translator used inside download() |
| *download_text*(url) | |

## Attributes

| |
|---|
| *CHUNK_SIZE* |

**CHUNK_SIZE**

**disable_ssl_verify_warning**()

**download**(*url*, *target_full_path*, *md5=None*, *sha256=None*, *size=None*, *progress_update_callback=None*)

**download_inner**(*url*, *target_full_path*, *md5*, *sha256*, *size*, *progress_update_callback*)

**download_partial_file**(*target_full_path: str | pathlib.Path*, *, *url: str*, *sha256: str*, *md5: str*, *size: int*)

Create or open locked partial download file, moving onto target_full_path when finished. Preserve partial file on exception.

**download_http_errors**(*url: str*)

Exception translator used inside download()

**download_text**(*url*)

**class TmpDownload**(*url*, *verbose=True*)

Context manager to handle downloads to a tempfile.

**__enter__**()

**__exit__**(*exc_type*, *exc_value*, *traceback*)

### session

Requests session configured with all accepted scheme adapters.

## Classes

| | |
|---|---|
| *EnforceUnusedAdapter* | The Base Transport Adapter |
| *CondaSessionType* | Takes advice from https://github.com/requests/requests/issues/1871#issuecomment-33327847 |
| *CondaSession* | A Requests session. |
| *CondaHttpAuth* | Base class that all auth implementations derive from |

## Functions

| | |
|---|---|
| *get_channel_name_from_url*(→ str \| None) | Given a URL, determine the channel it belongs to and return its name. |
| *get_session*(url) | Function that determines the correct Session object to be returned |
| *get_session_storage_key*(→ str) | Function that determines which storage key to use for our CondaSession object caching |

## Attributes

| |
|---|
| *RETRIES* |
| *CONDA_SESSION_SCHEMES* |

**RETRIES = 3**

**CONDA_SESSION_SCHEMES**

class EnforceUnusedAdapter

Bases: conda.gateways.connection.BaseAdapter

The Base Transport Adapter

send(*request: requests.models.Request*, **args*, ***kwargs*)

Sends PreparedRequest object. Returns Response object.

**Parameters**

- **request** -- The PreparedRequest being sent.
- **stream** -- (optional) Whether to stream the request content.
- **timeout** (*float or tuple*) -- (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **verify** -- (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- (optional) Any user-provided SSL certificate to be trusted.
- **proxies** -- (optional) The proxies dictionary to apply to the request.

abstract close()

Cleans up adapter specific items.

get_channel_name_from_url(*url: str*) → str | None

Given a URL, determine the channel it belongs to and return its name.

get_session(*url: str*)

Function that determines the correct Session object to be returned based on the URL that is passed in.

get_session_storage_key(*auth*) → str

Function that determines which storage key to use for our CondaSession object caching

class CondaSessionType

Bases: type

Takes advice from https://github.com/requests/requests/issues/1871#issuecomment-33327847 and creates one Session instance per thread.

__call__(***kwargs*)

Call self as a function.

class CondaSession(*auth: conda.gateways.connection.AuthBase | tuple[str, str] | None = None*)

Bases: conda.gateways.connection.Session

A Requests session.

Provides cookie persistence, connection-pooling, and configuration.

Basic Usage:

```
>>> import requests
>>> s = requests.Session()
>>> s.get('https://httpbin.org/get')
<Response [200]>
```

Or as a context manager:

```
>>> with requests.Session() as s:
...     s.get('https://httpbin.org/get')
<Response [200]>
```

**Parameters**
**auth** -- Optionally provide `requests.AuthBase` compliant objects

**prepare_request**(*request: requests.models.Request*) → requests.models.PreparedRequest

Constructs a `PreparedRequest` for transmission and returns it. The `PreparedRequest` has settings merged from the `Request` instance and those of the `Session`.

**Parameters**
**request** -- `Request` instance to prepare with this session's settings.

**Return type**
requests.PreparedRequest

**classmethod cache_clear**()

**class CondaHttpAuth**

Bases: `conda.gateways.connection.AuthBase`

Base class that all auth implementations derive from

**__call__**(*request*)

**static _apply_basic_auth**(*request*)

**static add_binstar_token**(*url*)

**static handle_407**(*response*, ***kwargs*)

Prompts the user for the proxy username and password and modifies the proxy in the session object to include it.

**This method is modeled after**

- requests.auth.HTTPDigestAuth.handle_401()
- requests.auth.HTTPProxyAuth
- the previous conda.fetch.handle_proxy_407()

It both adds 'username:password' to the proxy URL, as well as adding a 'Proxy-Authorization' header. If any of this is incorrect, please file an issue.

### disk

### create

Disk utility functions for creating new files or directories.

### Classes

| | |
|---|---|
| *TemporaryDirectory* | Create and return a temporary directory. This has the same |
| *ProgressFileWrapper* | |

### Functions

| | |
|---|---|
| *write_as_json_to_file*(file_path, obj) | |
| *create_python_entry_point*(target_full_path, ...) | |
| *create_application_entry_point*(source_full_path ...) | |
| *extract_tarball*(tarball_full_path[, ...]) | |
| *make_menu*(prefix, file_path[, remove]) | Create cross-platform menu items (e.g. Windows Start Menu) |
| *create_hard_link_or_copy*(src, dst) | |
| *_is_unix_executable_using_ORIGIN*(path) | |
| *_do_softlink*(src, dst) | |
| *create_fake_executable_softlink*(src, dst) | |
| *copy*(src, dst) | |
| *_do_copy*(src, dst) | |
| *create_link*(src, dst[, link_type, force]) | |
| *compile_multiple_pyc*(python_exe_full_path, ...) | |
| *create_package_cache_directory*(pkgs_dir) | |
| *create_envs_directory*(envs_dir) | |
| *first_writable_envs_dir*([create]) | |

### Attributes

| |
|---|
| *stdoutlog* |
| *mkdir_p* |
| *python_entry_point_template* |
| *application_entry_point_template* |

**class TemporaryDirectory**(*suffix=''*, *prefix='tmp'*, *dir=None*)

Create and return a temporary directory. This has the same behavior as mkdtemp but can be used as a context manager. For .. rubric:: Example

**with TemporaryDirectory() as tmpdir:**

...

Upon exiting the context, the directory and everything contained in it are removed.

**name**

**_closed = False**

**__repr__**()

Return repr(self).

**__enter__**()

**cleanup**(*_warn=False*, *_warnings=_warnings*)

**__exit__**(*exc*, *value*, *tb*)

**__del__**()

**stdoutlog**

**mkdir_p**

**python_entry_point_template**

**application_entry_point_template**

**write_as_json_to_file**(*file_path*, *obj*)

**create_python_entry_point**(*target_full_path*, *python_full_path*, *module*, *func*)

**create_application_entry_point**(*source_full_path*, *target_full_path*, *python_full_path*)

**class ProgressFileWrapper**(*fileobj*, *progress_update_callback*)

**__getattr__**(*name*)

**__setattr__**(*name*, *value*)

Implement setattr(self, name, value).

**read**(*size=-1*)

**progress_update**()

**extract_tarball**(*tarball_full_path*, *destination_directory=None*, *progress_update_callback=None*)

**make_menu**(*prefix*, *file_path*, *remove=False*)

Create cross-platform menu items (e.g. Windows Start Menu)

Passes all menu config files %PREFIX%/Menu/*.json to `menuinst.install`. `remove=True` will remove the menu items.

**create_hard_link_or_copy**(*src*, *dst*)

**_is_unix_executable_using_ORIGIN**(*path*)

**_do_softlink**(*src*, *dst*)

**create_fake_executable_softlink**(*src*, *dst*)

**copy**(*src*, *dst*)

**_do_copy**(*src*, *dst*)

**create_link**(*src*, *dst*, *link_type=LinkType.hardlink*, *force=False*)

**compile_multiple_pyc**(*python_exe_full_path*, *py_full_paths*, *pyc_full_paths*, *prefix*, *py_ver*)

**create_package_cache_directory**(*pkgs_dir*)

**create_envs_directory**(*envs_dir*)

**first_writable_envs_dir**(*create=True*)

### delete

Disk utility functions for deleting files and folders.

#### Functions

| | |
|---|---|
| *rmtree*(path) | |
| *unlink_or_rename_to_trash*(path) | If files are in use, especially on windows, we can't remove them. |
| *remove_empty_parent_paths*(path) | |
| *rm_rf*(→ bool) | Completely delete path |
| *delete_trash*(prefix) | |
| *backoff_rmdir*(dirpath[, max_tries]) | |
| *path_is_clean*(path) | Sometimes we can't completely remove a path because files are considered in use |

**rmtree**(*path*)

**unlink_or_rename_to_trash**(*path*)

If files are in use, especially on windows, we can't remove them. The fallback path is to rename them (but keep their folder the same), which maintains the file handle validity. See comments at: https://serverfault.com/a/503769

**remove_empty_parent_paths**(*path*)

**rm_rf**(*path: str | os.PathLike*, *clean_empty_parents: bool = False*) → bool

Completely delete path max_retries is the number of times to retry on failure. The default is 5. This only applies to deleting a directory. If removing path fails and trash is True, files will be moved to the trash directory.

**delete_trash**(*prefix*)

**backoff_rmdir**(*dirpath*, *max_tries=MAX_TRIES*)

**path_is_clean**(*path*)

Sometimes we can't completely remove a path because files are considered in use by python (hardlinking confusion). For our tests, it is sufficient that either the folder doesn't exist, or nothing but temporary file copies are left.

### link

Disk utility functions for symlinking files and folders.

Portions of the code within this module are taken from https://github.com/jaraco/jaraco.windows which is MIT licensed by Jason R. Coombs.

https://github.com/jaraco/skeleton/issues/1#issuecomment-285448440

**lchmod**

**link**

**islink**

### lock

Record locking to manage potential repodata / repodata metadata file contention between conda processes. Try to acquire a lock on a single byte in the metadat file; modify both files; then release the lock.

## Functions

| | |
|---|---|
| *_lock_noop*(fd) | When locking is not available. |
| *_lock_impl*(fd) | |
| *lock*(fd) | |

### Attributes

| |
|---|
| *LOCK_BYTE* |
| *LOCK_ATTEMPTS* |
| *LOCK_SLEEP* |

**LOCK_BYTE = 21**

**LOCK_ATTEMPTS = 10**

**LOCK_SLEEP = 1**

**_lock_noop**(*fd*)

When locking is not available.

**_lock_impl**(*fd*)

**lock**(*fd*)

## permissions

Disk utility functions for modifying file and directory permissions.

### Functions

| |
|---|
| *make_writable*(path) |
| *make_read_only*(path) |
| *recursive_make_writable*(path[, max_tries]) |
| *make_executable*(path) |
| *is_executable*(path) |

**make_writable**(*path*)

**make_read_only**(*path*)

**recursive_make_writable**(*path*, *max_tries=MAX_TRIES*)

**make_executable**(*path*)

**is_executable**(*path*)

### read

Disk utility functions for reading and processing file contents.

## Functions

| | |
|---|---|
| *yield_lines*(path) | Generator function for lines in file. Empty generator if path does not exist. |
| *compute_sum*(→ str) | |
| *read_package_info*(record, package_cache_record) | |
| *read_index_json*(extracted_package_directory) | |
| *read_index_json_from_tarball*(package_tarball_ful | |
| *read_repodata_json*(extracted_package_directory) | |
| *read_icondata*(extracted_package_directory) | |
| *read_package_metadata*(extracted_package_directory | |
| *read_paths_json*(extracted_package_directory) | |
| *read_has_prefix*(path) | Reads *has_prefix* file and return dict mapping filepaths to tuples(placeholder, FileMode). |
| *read_no_link*(info_dir) | |
| *read_soft_links*(extracted_package_directory, files) | |
| *read_python_record*(prefix_path, anchor_file, ...) | |

## Attributes

| |
|---|
| *listdir* |

### listdir

### yield_lines(*path*)

Generator function for lines in file. Empty generator if path does not exist.

**Parameters**
**path** (*str*) -- path to file

**Returns**
each line in file, not starting with '#'

**Return type**
iterator

**compute_sum**(*path: str | os.PathLike*, *algo: Literal[md5, sha256]*) → str

**read_package_info**(*record*, *package_cache_record*)

**read_index_json**(*extracted_package_directory*)

**read_index_json_from_tarball**(*package_tarball_full_path*)

**read_repodata_json**(*extracted_package_directory*)

**read_icondata**(*extracted_package_directory*)

**read_package_metadata**(*extracted_package_directory*)

**read_paths_json**(*extracted_package_directory*)

**read_has_prefix**(*path*)

Reads *has_prefix* file and return dict mapping filepaths to tuples(placeholder, FileMode).

**A line in *has_prefix* contains one of:**

- filepath
- placeholder mode filepath

**Mode values are one of:**

- text
- binary

**read_no_link**(*info_dir*)

**read_soft_links**(*extracted_package_directory*, *files*)

**read_python_record**(*prefix_path*, *anchor_file*, *python_version*)

### test

Disk utility functions testing path properties (e.g., writable, hardlinks, softlinks, etc.).

#### Functions

| | |
|---|---|
| *file_path_is_writable*(→ bool) | |
| *hardlink_supported*(source_file, dest_dir) | |
| *softlink_supported*(source_file, dest_dir) | |
| *is_conda_environment*(prefix) | |

**file_path_is_writable**(*path*) → bool

**hardlink_supported**(*source_file*, *dest_dir*)

**softlink_supported**(*source_file*, *dest_dir*)

**is_conda_environment**(*prefix*)

#### update

Disk utility functions for modifying existing files or directories.

### Functions

| | |
|---|---|
| *update_file_in_place_as_binary*(file_full_path, callback) | |
| *rename*(source_path, destination_path[, force]) | |
| *rename_context*(source[, destination, dry_run]) | Used for removing a directory when there are dependent actions (i.e. you need to ensure |
| *backoff_rename*(source_path, destination_path[, force]) | |
| *touch*(path[, mkdir, sudo_safe]) | |

### Attributes

| |
|---|
| *SHEBANG_REGEX* |

**SHEBANG_REGEX**

**exception CancelOperation**

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**update_file_in_place_as_binary**(*file_full_path*, *callback*)

**rename**(*source_path*, *destination_path*, *force=False*)

**rename_context**(*source: str*, *destination: str | None = None*, *dry_run: bool = False*)

Used for removing a directory when there are dependent actions (i.e. you need to ensure other actions succeed before removing it).

#### Example

**with rename_context(directory):**
    # Do dependent actions here

**backoff_rename**(*source_path*, *destination_path*, *force=False*)

**touch**(*path*, *mkdir=False*, *sudo_safe=False*)

### Functions

| | |
|---|---|
| *exp_backoff_fn*(fn, *args, **kwargs) | Mostly for retrying file operations that fail on Windows due to virus scanners |
| *mkdir_p*(path) | |
| *mkdir_p_sudo_safe*(path) | |

### Attributes

| |
|---|
| *on_win* |
| *TRACE* |
| *MAX_TRIES* |

**on_win**

**TRACE = 5**

**MAX_TRIES = 7**

**exp_backoff_fn**(*fn*, **args*, ***kwargs*)

Mostly for retrying file operations that fail on Windows due to virus scanners

**mkdir_p**(*path*)

**mkdir_p_sudo_safe**(*path*)

## logging

Configure logging for conda.

### Classes

| | |
|---|---|
| *TokenURLFilter* | Filter instances are used to perform arbitrary filtering of LogRecords. |
| *StdStreamHandler* | Log StreamHandler that always writes to the current sys stream. |

### Functions

| |
|---|
| *initialize_logging*() |
| *initialize_std_loggers*() |
| *initialize_root_logger*([level]) |
| *set_conda_log_level*([level]) |
| *set_all_logger_level*([level]) |
| *set_file_logging*([logger_name, level, path]) |
| *set_log_level*(log_level) |

### Attributes

| |
|---|
| *_VERBOSITY_LEVELS* |

**_VERBOSITY_LEVELS**

**class TokenURLFilter**(*name=''*)

Bases: logging.Filter

Filter instances are used to perform arbitrary filtering of LogRecords.

Loggers and Handlers can optionally use Filter instances to filter records as desired. The base filter class only allows events which are below a certain point in the logger hierarchy. For example, a filter initialized with "A.B" will allow events logged by loggers "A.B", "A.B.C", "A.B.C.D", "A.B.D" etc. but not "A.BB", "B.A.B" etc. If initialized with the empty string, all events are passed.

Initialize a filter.

Initialize with the name of the logger which, together with its children, will have its events allowed through the filter. If no name is specified, allow every event.

**TOKEN_URL_PATTERN**

**TOKEN_REPLACE**

**filter**(*record*)

Since Python 2's getMessage() is incapable of handling any strings that are not unicode when it interpolates the message with the arguments, we fix that here by doing it ourselves.

At the same time we replace tokens in the arguments which was not happening until now.

**class StdStreamHandler**(*sys_stream*)

Bases: logging.StreamHandler

Log StreamHandler that always writes to the current sys stream.

**Parameters**
**sys_stream** -- stream name, either "stdout" or "stderr" (attribute of module sys)

**terminator = '\n'**

**__getattr__**(*attr*)

**emit**(*record*)

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

**initialize_logging**()

**initialize_std_loggers**()

**initialize_root_logger**(*level=ERROR*)

**set_conda_log_level**(*level=WARN*)

**set_all_logger_level**(*level=DEBUG*)

**set_file_logging**(*logger_name=None*, *level=DEBUG*, *path=None*)

**set_log_level**(*log_level: int*)

### repodata

Repodata interface.

### jlap

Incremental repodata feature based on .jlap patch files.

### core

JLAP reader.

## Classes

| | |
|---|---|
| *JLAP* | A more or less complete user-defined wrapper around list objects. |

## Functions

| | |
|---|---|
| *keyed_hash*(data, key) | Keyed hash. |
| *line_and_pos*(→ collections.abc.Iterator[tuple[int, bytes]]) | |

## Attributes

| |
|---|
| *DIGEST_SIZE* |
| *DEFAULT_IV* |

**DIGEST_SIZE = 32**

**DEFAULT_IV**

**keyed_hash**(*data: bytes*, *key: bytes*)

Keyed hash.

**line_and_pos**(*lines: collections.abc.Iterable[bytes]*, *pos=0*) → collections.abc.Iterator[tuple[int, bytes]]

**Parameters**

- **lines** -- iterator over input split by 'n', with 'n' removed.
- **pos** -- initial position

**class JLAP**(*initlist=None*)

Bases: collections.UserList

A more or less complete user-defined wrapper around list objects.

**property body**

All lines except the first, and last two.

**property penultimate**

Next-to-last line. Should contain the footer.

**property last**

Last line. Should contain the trailing checksum.

**classmethod from_lines**(*lines: collections.abc.Iterable[bytes]*, *iv: bytes*, *pos=0*, *verify=True*)

**Parameters**

- **lines** -- iterator over input split by b'n', with b'n' removed
- **pos** -- initial position
- **iv** -- initialization vector (first line of .jlap stream, hex decoded). Ignored if pos==0.
- **verify** -- assert last line equals computed checksum of previous line. Useful for writing new .jlap files if False.

**Raises**

**ValueError** -- if trailing and computed checksums do not match

**Returns**

list of (offset, line, checksum)

**classmethod from_path**(*path: pathlib.Path | str*, *verify=True*)

**add**(*line: str*)

Add line to buffer, following checksum rules.

Buffer must not be empty.

(Remember to pop trailing checksum and possibly trailing metadata line, if appending to a complete jlap file)

Less efficient than creating a new buffer from many lines and our last iv, and extending.

**Returns**

self

**terminate**()

Add trailing checksum to buffer.

**Returns**

self

**write**(*path: pathlib.Path*)

Write buffer to path.

### fetch

JLAP consumer.

### Classes

| | |
|---|---|
| *HashWriter* | Base class for raw binary I/O. |

### Functions

| | |
|---|---|
| *hash*() | Ordinary hash. |
| *process_jlap_response*(response[, pos, iv]) | |
| *fetch_jlap*(url[, pos, etag, iv, ignore_etag, session]) | |
| *request_jlap*(url[, pos, etag, ignore_etag, session]) | Return the part of the remote .jlap file we are interested in. |
| *format_hash*(hash) | Abbreviate hash for formatting. |
| *find_patches*(patches, have, want) | |
| *apply_patches*(data, apply) | |
| *withext*(url, ext) | |
| *timeme*(message) | |
| *build_headers*(json_path, state) | Caching headers for a path and state. |
| *download_and_hash*(hasher, url, json_path, session, state) | Download url if it doesn't exist, passing bytes through hasher.update(). |
| *_is_http_error_most_400_codes*(→ bool) | Determine whether the *HTTPError* is an HTTP 400 error code (except for 416). |
| *request_url_jlap_state*(→ dict \| None) | |

### Attributes

| |
|---|
| *DIGEST_SIZE* |
| *JLAP_KEY* |
| *HEADERS* |
| *NOMINAL_HASH* |
| *ON_DISK_HASH* |
| *LATEST* |
| *STORE_HEADERS* |

DIGEST_SIZE = 32

JLAP_KEY = 'jlap'

HEADERS = 'headers'

NOMINAL_HASH = 'blake2_256_nominal'

ON_DISK_HASH = 'blake2_256'

LATEST = 'latest'

STORE_HEADERS

hash()

Ordinary hash.

exception Jlap304NotModified

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception JlapSkipZst

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

exception JlapPatchNotFound

Bases: LookupError

Base class for lookup errors.

Initialize self. See help(type(self)) for accurate signature.

process_jlap_response(*response: conda.gateways.connection.Response*, *pos=0*, *iv=b''*)

fetch_jlap(*url*, *pos=0*, *etag=None*, *iv=b''*, *ignore_etag=True*, *session=None*)

request_jlap(*url*, *pos=0*, *etag=None*, *ignore_etag=True*, *session: conda.gateways.connection.Session | None = None*)

Return the part of the remote .jlap file we are interested in.

format_hash(*hash*)

Abbreviate hash for formatting.

find_patches(*patches*, *have*, *want*)

apply_patches(*data*, *apply*)

withext(*url*, *ext*)

timeme(*message*)

build_headers(*json_path: pathlib.Path*, *state:* conda.gateways.repodata.RepodataState)

Caching headers for a path and state.

**class HashWriter**(*backing*, *hasher*)

Bases: io.RawIOBase

Base class for raw binary I/O.

Initialize self. See help(type(self)) for accurate signature.

**write**(*b: bytes*)

**close**()

Flush and close the IO object.

This method has no effect if the file is already closed.

**download_and_hash**(*hasher*, *url*, *json_path: pathlib.Path*, *session: conda.gateways.connection.Session*, *state:* conda.gateways.repodata.RepodataState | *None*, *is_zst=False*, *dest_path: pathlib.Path* | *None* = *None*)

Download url if it doesn't exist, passing bytes through hasher.update().

json_path: Path of old cached data (ignore etag if not exists). dest_path: Path to write new data.

**_is_http_error_most_400_codes**(*e: requests.HTTPError*) → bool

Determine whether the *HTTPError* is an HTTP 400 error code (except for 416).

**request_url_jlap_state**(*url*, *state:* conda.gateways.repodata.RepodataState, *full_download=False*, ***, *session: conda.gateways.connection.Session*, *cache:* conda.gateways.repodata.RepodataCache, *temp_path: pathlib.Path*) → dict | None

### interface

JLAP interface for repodata.

## Classes

| | |
|---|---|
| *JlapRepoInterface* | Helper class that provides a standard way to create an ABC using |
| *RepodataStateSkipFormat* | Load/save info file that accompanies cached *repodata.json*. |
| *ZstdRepoInterface* | Support repodata.json.zst (if available) without checking .jlap |

**class JlapRepoInterface**(*url: str*, *repodata_fn: str* | *None*, ***, *cache:* conda.gateways.repodata.RepodataCache, ***kwargs*)

Bases: *conda.gateways.repodata.RepoInterface*

Helper class that provides a standard way to create an ABC using inheritance.

**repodata**(*state: dict* | conda.gateways.repodata.RepodataState) → str | None

Fetch newest repodata if necessary.

Always writes to `cache_path_json`.

**repodata_parsed**(*state: dict* | conda.gateways.repodata.RepodataState) → dict | None

JLAP has to parse the JSON anyway.

Use this to avoid a redundant parse when repodata is updated.

When repodata is not updated, it doesn't matter whether this function or the caller reads from a file.

**_repodata_state_copy**(*state: dict* | conda.gateways.repodata.RepodataState)

**class RepodataStateSkipFormat**(**args*, *skip_formats=set()*, ***kwargs*)

Bases: *conda.gateways.repodata.RepodataState*

Load/save info file that accompanies cached *repodata.json*.

**skip_formats:** set[str]

**should_check_format**(*format*)

Return True if named format should be attempted.

**class ZstdRepoInterface**(*url: str*, *repodata_fn: str* | *None*, ***, *cache:* conda.gateways.repodata.RepodataCache, ***kwargs*)

Bases: *JlapRepoInterface*

Support repodata.json.zst (if available) without checking .jlap

**_repodata_state_copy**(*state: dict* | conda.gateways.repodata.RepodataState)

### lock

Backwards compatibility import.

Moved to prevent circular imports.

## Classes

| | |
|---|---|
| *PackageCacheData* | |
| *Channel* | Channel: |
| *RepoInterface* | Helper class that provides a standard way to create an ABC using |
| *CondaRepoInterface* | Provides an interface for retrieving repodata data from channels. |
| *RepodataState* | Load/save info file that accompanies cached *repodata.json*. |
| *RepodataCache* | Handle caching for a single repodata.json + repodata.info.json |
| *RepodataFetch* | Combine RepodataCache and RepoInterface to provide subdir_data.SubdirData() |

### Functions

| | |
|---|---|
| *stringify*(obj[, content_max_len]) | |
| *maybe_unquote*(→ str) | |
| *get_session*(url) | Function that determines the correct Session object to be returned |
| *mkdir_p_sudo_safe*(path) | |
| *lock*(fd) | |
| *get_repo_interface*(→ type[RepoInterface]) | |
| *_add_http_value_to_dict*(resp, http_key, d, dict_key) | |
| *conda_http_errors*(url, repodata_fn) | Use in a with: statement to translate requests exceptions to conda ones. |
| *_md5_not_for_security*(data) | |
| *cache_fn_url*(url[, repodata_fn]) | |
| *get_cache_control_max_age*(cache_control_value) | |
| *create_cache_dir*() | |

### Attributes

| |
|---|
| *CONDA_HOMEPAGE_URL* |
| *REPODATA_FN* |
| *context* |
| *join_url* |
| *stderrlog* |
| *CHECK_ALTERNATE_FORMAT_INTERVAL* |
| *LAST_MODIFIED_KEY* |
| *ETAG_KEY* |
| *CACHE_CONTROL_KEY* |
| *URL_KEY* |
| *CACHE_STATE_SUFFIX* |
| *ERROR_SNIPPET_LENGTH* |

**exception CondaError**(*message: str | None*, *caused_by: Any = None*, ***kwargs*)

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**return_code: int = 1**

**reportable: bool = False**

**__repr__**() → str

Return repr(self).

**__str__**() → str

Return str(self).

**dump_map**() → dict[str, Any]

**stringify**(*obj*, *content_max_len=0*)

**CONDA_HOMEPAGE_URL: Final = 'https://conda.io'**

**REPODATA_FN: Final = 'repodata.json'**

**context**

**join_url**

**maybe_unquote**(*url: str*) → str

**class PackageCacheData**(*pkgs_dir*)

**property _package_cache_records**

**property is_writable**

**_cache_: dict[str, PackageCacheData]**

**insert**(*package_cache_record*)

**load**()

**reload**()

**get**(*package_ref*, *default=NULL*)

**remove**(*package_ref*, *default=NULL*)

**query**(*package_ref_or_match_spec*)

**iter_records**()

**classmethod query_all**(*package_ref_or_match_spec*, *pkgs_dirs=None*)

**classmethod first_writable**(*pkgs_dirs=None*)

**classmethod writable_caches**(*pkgs_dirs=None*)

**classmethod read_only_caches**(*pkgs_dirs=None*)

**classmethod all_caches_writable_first**(*pkgs_dirs=None*)

**classmethod get_all_extracted_entries**()

**classmethod get_entry_to_link**(*package_ref*)

**classmethod tarball_file_in_cache**(*tarball_path*, *md5sum=None*, *exclude_caches=()*)

**classmethod clear**()

**tarball_file_in_this_cache**(*tarball_path*, *md5sum=None*)

**_check_writable**()

**static _clean_tarball_path_and_get_md5sum**(*tarball_path*, *md5sum=None*)

**_scan_for_dist_no_channel**(*dist_str*)

**itervalues**()

**values**()

**__repr__**()

Return repr(self).

**_make_single_record**(*package_filename*)

**static _dedupe_pkgs_dir_contents**(*pkgs_dir_contents*)

**exception CondaDependencyError**(*message: str*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaHTTPError**(*message: str*, *url: str*, *status_code: int | str*, *reason: str*, *elapsed_time: datetime.timedelta | str*, *response: requests.Response | None = None*, *caused_by: Any = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception CondaSSLError**(*message: str | None*, *caused_by: Any | None = None*, ***kwargs*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception NotWritableError**(*path: os.PathLike*, *errno: int*, ***kwargs*)

Bases: *conda.CondaError*, OSError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception ProxyError**(*message: str | None = None*)

Bases: *conda.CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**exception UnavailableInvalidChannel**(*channel:* conda.models.channel.Channel *| str*, *status_code: str | int*, *response: requests.models.Response | None = None*)

Bases: ChannelError

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**status_code: str | int**

**class Channel**(*scheme: str | None = None*, *auth: str | None = None*, *location: str | None = None*, *token: str | None = None*, *name: str | None = None*, *platform: str | None = None*, *package_filename: str | None = None*)

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

**property channel_location: str | None**

**property channel_name: str**

**property subdir: str | None**

**property canonical_name: str**

**property base_url:** `str | None`

**property base_urls:** `tuple[str | None]`

**property subdir_url:** `str`

**property url_channel_wtf:** `tuple[str | None, str]`

**_cache_**

**static _reset_state**() → None

**static from_url**(*url: str*) → *Channel*

**static from_channel_name**(*channel_name: str*) → *Channel*

**static from_value**(*value: str | None*) → *Channel*

Construct a new *Channel* from a single value.

**Parameters**

**value** -- Anyone of the following forms:

***None*, or one of the special strings "<unknown>", "None:///<unknown>", or "None":**
represents the unknown channel, used for packages with unknown origin.

**A URL including a scheme like `file://` or `https://`:**
represents a channel URL.

**A local directory path:**
represents a local channel; relative paths must start with `./`.

**A package file (i.e. the path to a file ending in `.conda` or `.tar.bz2`):**
represents a channel for a single package

**A known channel name:**
represents a known channel, e.g. from the users `.condarc` file or the global configuration.

**Returns**

A channel object.

**static make_simple_channel**(*channel_alias: Channel*, *channel_url: str*, *name: str | None = None*) → *Channel*

**urls**(*with_credentials: bool = False*, *subdirs: collections.abc.Iterable[str] | None = None*) → list[str]

**url**(*with_credentials: bool = False*) → str | None

**__str__**() → str

Return str(self).

**__repr__**() → str

Return repr(self).

**__eq__**(*other: Any*) → bool

Return self==value.

**__hash__**() → int

Return hash(self).

**__nonzero__**() → bool

__bool__() → bool

__json__() → dict[str, Any]

dump() → dict[str, str | None]

get_session(*url: str*)

Function that determines the correct Session object to be returned based on the URL that is passed in.

mkdir_p_sudo_safe(*path*)

lock(*fd*)

stderrlog

CHECK_ALTERNATE_FORMAT_INTERVAL

LAST_MODIFIED_KEY = 'mod'

ETAG_KEY = 'etag'

CACHE_CONTROL_KEY = 'cache_control'

URL_KEY = 'url'

CACHE_STATE_SUFFIX = '.info.json'

ERROR_SNIPPET_LENGTH = 32

exception RepodataIsEmpty(*channel: conda.models.channel.Channel | str, status_code: str | int, response: requests.models.Response | None = None*)

Bases: *conda.exceptions.UnavailableInvalidChannel*

Subclass used to determine when empty repodata should be cached, e.g. for a channel that doesn't provide current_repodata.json

Initialize self. See help(type(self)) for accurate signature.

exception RepodataOnDisk

Bases: Exception

Indicate that RepoInterface.repodata() successfully wrote repodata to disk, instead of returning a string.

Initialize self. See help(type(self)) for accurate signature.

class RepoInterface

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

repodata(*state: dict*) → str

Given a mutable state dictionary with information about the cache, return repodata.json (or current_repodata.json) as a str. This function also updates state, which is expected to be saved by the caller.

exception Response304ContentUnchanged

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**get_repo_interface**() → type[*RepoInterface*]

**class CondaRepoInterface**(*url: str*, *repodata_fn: str | None*, ***kwargs*)

Bases: *RepoInterface*

Provides an interface for retrieving repodata data from channels.

**_url: str**

**_repodata_fn: str**

**repodata**(*state:* RepodataState) → str | None

Given a mutable state dictionary with information about the cache, return repodata.json (or current_repodata.json) as a str. This function also updates state, which is expected to be saved by the caller.

**_add_http_value_to_dict**(*resp*, *http_key*, *d*, *dict_key*)

**conda_http_errors**(*url*, *repodata_fn*)

Use in a with: statement to translate requests exceptions to conda ones.

**class RepodataState**(*cache_path_json: pathlib.Path | str = ''*, *cache_path_state: pathlib.Path | str = ''*, *repodata_fn=''*, *dict=None*)

Bases: collections.UserDict

Load/save info file that accompanies cached *repodata.json*.

**property mod: str**

Last-Modified header or ""

**property etag: str**

Etag header or ""

**property cache_control: str**

Cache-Control header or ""

**_aliased**

**_strings**

**has_format**(*format: str*) → tuple[bool, datetime.datetime | None]

**set_has_format**(*format: str*, *value: bool*)

**clear_has_format**(*format: str*)

Remove 'has_{format}' instead of setting to False.

**should_check_format**(*format: str*) → bool

Return True if named format should be attempted.

**__contains__**(*key: str*) → bool

**__setitem__**(*key: str*, *item: Any*) → None

**__getitem__**(*key: str*) → Any

**class RepodataCache**(*base*, *repodata_fn*)

Handle caching for a single repodata.json + repodata.info.json (<hex-string>*.json inside *dir*)

Avoid race conditions while loading, saving repodata.json and cache state.

base: directory and filename prefix for cache, e.g. /cache/dir/abc123; writes /cache/dir/abc123.json

**property cache_path_json**

**property cache_path_state**

Out-of-band etag and other state needed by the RepoInterface.

**load**(*, *state_only=False*) → str

**load_state**()

Update self.state without reading repodata.json.

Return self.state.

**save**(*data: str*)

Write data to <repodata>.json cache path, synchronize state.

**replace**(*temp_path: pathlib.Path*)

Rename path onto <repodata>.json path, synchronize state.

Relies on path's mtime not changing on move. *temp_path* should be adjacent to *self.cache_path_json* to be on the same filesystem.

**refresh**(*refresh_ns=0*)

Update access time in cache info file to indicate a HTTP 304 Not Modified response.

**lock**(*mode='a+'*)

Lock .info.json file. Hold lock while modifying related files.

mode: "a+" then seek(0) to write/create; "r+" to read.

**stale**()

Compare state refresh_ns against cache control header and context.local_repodata_ttl.

**timeout**()

Return number of seconds until cache times out (<= 0 if already timed out).

**class RepodataFetch**(*cache_path_base: pathlib.Path*, *channel:* conda.models.channel.Channel, *repodata_fn: str*, *, *repo_interface_cls*)

Combine RepodataCache and RepoInterface to provide subdir_data.SubdirData() with what it needs.

Provide a variety of formats since some `RepoInterface` have to `json.loads(...)` anyway, and some clients don't need the Python data structure at all.

**property url_w_repodata_fn**

**property cache_path_json**

**property cache_path_state**

Out-of-band etag and other state needed by the RepoInterface.

**property repo_cache:** *RepodataCache*

**property _repo:** *RepoInterface*

Changes as we mutate self.repodata_fn.

**cache_path_base:** pathlib.Path

**channel:** *conda.models.channel.Channel*

**repodata_fn:** str

**url_w_subdir:** str

**url_w_credentials:** str

**repo_interface_cls:** Any

**fetch_latest_parsed**() → tuple[dict, *RepodataState*]

Retrieve parsed latest or latest-cached repodata as a dict; update cache.

**Returns**
(repodata contents, state including cache headers)

**fetch_latest_path**() → tuple[pathlib.Path, *RepodataState*]

Retrieve latest or latest-cached repodata; update cache.

**Returns**
(pathlib.Path to uncompressed repodata contents, RepodataState)

**fetch_latest**() → tuple[dict | str, *RepodataState*]

Return up-to-date repodata and cache information. Fetch repodata from remote if cache has expired; return cached data if cache has not expired; return stale cached data or dummy data if in offline mode.

**read_cache**() → tuple[str, *RepodataState*]

Read repodata from disk, without trying to fetch a fresh version.

**_md5_not_for_security**(*data*)

**cache_fn_url**(*url*, *repodata_fn=REPODATA_FN*)

**get_cache_control_max_age**(*cache_control_value: str | None*)

**create_cache_dir**()

### subprocess

Helpler functions for subprocess.

## Functions

| | |
|---|---|
| *_format_output*(command_str, cwd, rc, stdout, stderr) | |
| *any_subprocess*(args, prefix[, env, cwd]) | |
| *subprocess_call*(command[, env, path, stdin, ...]) | This utility function should be preferred for all conda subprocessing. |
| *_subprocess_clean_env*(env[, clean_python, clean_conda]) | |
| *subprocess_call_with_clean_env*(command[, path, stdin, ...]) | |

### Attributes

| |
|---|
| *Response* |

#### Response

**_format_output**(*command_str*, *cwd*, *rc*, *stdout*, *stderr*)

**any_subprocess**(*args*, *prefix*, *env=None*, *cwd=None*)

**subprocess_call**(*command: str | os.PathLike | pathlib.Path | collections.abc.Sequence[str | os.PathLike | pathlib.Path]*, *env: dict[str, str] | None = None*, *path: str | os.PathLike | pathlib.Path | None = None*, *stdin: str | None = None*, *raise_on_error: bool = True*, *capture_output: bool = True*)

This utility function should be preferred for all conda subprocessing. It handles multiple tricky details.

**_subprocess_clean_env**(*env*, *clean_python=True*, *clean_conda=True*)

**subprocess_call_with_clean_env**(*command*, *path=None*, *stdin=None*, *raise_on_error=True*, *clean_python=True*, *clean_conda=True*)

## history

Tools interfacing with conda's history file.

### Classes

| |
|---|
| *History* |

### Functions

| |
|---|
| *write_head*(fo) |
| *is_diff*(content) |
| *pretty_diff*(diff) |
| *pretty_content*(content) |

### Attributes

*h*

**exception CondaHistoryWarning**

Bases: Warning

Base class for warning categories.

Initialize self. See help(type(self)) for accurate signature.

**write_head**(*fo*)

**is_diff**(*content*)

**pretty_diff**(*diff*)

**pretty_content**(*content*)

**class History**(*prefix*)

**com_pat**

**spec_pat**

**conda_v_pat**

**__enter__**()

**__exit__**(*exc_type*, *exc_value*, *traceback*)

**init_log_file**()

**file_is_empty**()

**update**() → None

Update the history file (creating a new one if necessary).

**parse**() → list[tuple[str, set[str], list[str]]]

Parse the history file.

Return a list of tuples(datetime strings, set of distributions/diffs, comments).

Comments appearing before the first section header (e.g. `==> 2024-01-01 00:00:00 <==`) in the history file will be ignored.

**static _parse_old_format_specs_string**(*specs_string*)

Parse specifications string that use conda<4.5 syntax.

#### Examples

- "param >=1.5.1,<2.0'"
- "python>=3.5.1,jupyter >=1.0.0,<2.0,matplotlib >=1.5.1,<2.0"

**classmethod _parse_comment_line**(*line*)

Parse comment lines in the history file.

These lines can be of command type or action type.

#### Examples

- "# cmd: /scratch/mc3/bin/conda install -c conda-forge param>=1.5.1,<2.0"
- "# install specs: python>=3.5.1,jupyter >=1.0.0,<2.0,matplotlib >=1.5.1,<2.0"

**get_user_requests**()

Return a list of user requested items.

Each item is a dict with the following keys: 'date': the date and time running the command 'cmd': a list of argv of the actual command which was run 'action': install/remove/update 'specs': the specs being used

**get_requested_specs_map**()

**construct_states**()

Return a list of tuples(datetime strings, set of distributions).

**get_state**(*rev=-1*)

Return the state, i.e. the set of distributions, for a given revision.

Defaults to latest (which is the same as the current state when the log file is up-to-date).

Returns a list of dist_strs.

**print_log**()

**object_log**()

**write_changes**(*last_state*, *current_state*)

**write_specs**(*remove_specs=()*, *update_specs=()*, *neutered_specs=()*)

**h**

### instructions

Define the instruction set (constants) for conda operations.

### Functions

| |
|---|
| *PRINT_CMD*(state, arg) |
| *FETCH_CMD*(state, package_cache_entry) |
| *EXTRACT_CMD*(state, arg) |
| *PROGRESSIVEFETCHEXTRACT_CMD*(state, ...) |
| *UNLINKLINKTRANSACTION_CMD*(state, arg) |
| *check_files_in_package*(source_dir, files) |

### Attributes

| |
|---|
| *CHECK_FETCH* |
| *FETCH* |
| *CHECK_EXTRACT* |
| *EXTRACT* |
| *RM_EXTRACTED* |
| *RM_FETCHED* |
| *PRINT* |
| *PROGRESS* |
| *SYMLINK_CONDA* |
| *UNLINK* |
| *LINK* |
| *UNLINKLINKTRANSACTION* |
| *PROGRESSIVEFETCHEXTRACT* |
| *PROGRESS_COMMANDS* |
| *ACTION_CODES* |
| *commands* |
| *OP_ORDER* |

**CHECK_FETCH = 'CHECK_FETCH'**

**FETCH = 'FETCH'**

**CHECK_EXTRACT = 'CHECK_EXTRACT'**

**EXTRACT = 'EXTRACT'**

**RM_EXTRACTED = 'RM_EXTRACTED'**

**RM_FETCHED = 'RM_FETCHED'**

**PRINT = 'PRINT'**

**PROGRESS = 'PROGRESS'**

SYMLINK_CONDA = 'SYMLINK_CONDA'

UNLINK = 'UNLINK'

LINK = 'LINK'

UNLINKLINKTRANSACTION = 'UNLINKLINKTRANSACTION'

PROGRESSIVEFETCHEXTRACT = 'PROGRESSIVEFETCHEXTRACT'

PROGRESS_COMMANDS

ACTION_CODES = ()

PRINT_CMD(*state*, *arg*)

FETCH_CMD(*state*, *package_cache_entry*)

EXTRACT_CMD(*state*, *arg*)

PROGRESSIVEFETCHEXTRACT_CMD(*state*, *progressive_fetch_extract*)

UNLINKLINKTRANSACTION_CMD(*state*, *arg*)

check_files_in_package(*source_dir*, *files*)

commands

OP_ORDER = ()

### misc

Miscellaneous utility functions.

### Functions

| | |
|---|---|
| *conda_installed_files*(prefix[, exclude_self_build]) | Return the set of files which have been installed (using conda) into |
| *_match_specs_from_explicit*(...) | |
| *explicit*(→ None) | |
| *install_explicit_packages*(package_cache_records prefix) | Install a list of PackageRecords into a prefix |
| *_get_package_record_from_specs*(...) | Given a list of specs, find the corresponding PackageCacheRecord. If |
| *get_package_records_from_explicit*(...) | Given the lines from an explicit.txt, create the PackageRecords for each of the |
| *rel_path*(prefix, path[, windows_forward_slashes]) | |
| *walk_prefix*(prefix[, ignore_predefined_files, ...]) | Return the set of all files in a given prefix directory. |
| *untracked*(prefix[, exclude_self_build]) | Return (the set) of all untracked files for a given prefix. |
| *touch_nonadmin*(prefix) | Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows). |
| *clone_env*(prefix1, prefix2[, verbose, quiet, index_args]) | Clone existing prefix1 into new prefix2. |
| *_get_best_prec_match*(precs) | |

### Attributes

| |
|---|
| *url_pat* |

**conda_installed_files**(*prefix*, *exclude_self_build=False*)

Return the set of files which have been installed (using conda) into a given prefix.

**url_pat**

**_match_specs_from_explicit**(*specs: collections.abc.Iterable[str]*) → collections.abc.Iterable[*conda.models.match_spec.MatchSpec*]

**explicit**(*specs: collections.abc.Iterable[str]*, *prefix: str*, *verbose: bool = False*, *force_extract: bool = True*, *index: Any = None*, *requested_specs: collections.abc.Sequence[str] | None = None*) → None

**install_explicit_packages**(*package_cache_records: list[conda.models.records.PackageRecord]*, *prefix: str*, *requested_specs: collections.abc.Sequence[str] | None = None*)

Install a list of PackageRecords into a prefix

**_get_package_record_from_specs**(*specs: list[str]*) → collections.abc.Iterable[*conda.models.records.PackageCacheRecord*]

Given a list of specs, find the corresponding PackageCacheRecord. If some PackageCacheRecords are missing, raise an error.

**get_package_records_from_explicit**(*lines: list[str]*) → collections.abc.Iterable[*conda.models.records.PackageCacheRecord*]

Given the lines from an explicit.txt, create the PackageRecords for each of the specified packages. This may require downloading the package, if it does not already exist in the package cache.

**rel_path**(*prefix*, *path*, *windows_forward_slashes=True*)

**walk_prefix**(*prefix*, *ignore_predefined_files=True*, *windows_forward_slashes=True*)

Return the set of all files in a given prefix directory.

**untracked**(*prefix*, *exclude_self_build=False*)

Return (the set) of all untracked files for a given prefix.

**touch_nonadmin**(*prefix*)

Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows).

**clone_env**(*prefix1*, *prefix2*, *verbose=True*, *quiet=False*, *index_args=None*)

Clone existing prefix1 into new prefix2.

**_get_best_prec_match**(*precs*)

### models

Models are data transfer objects or "light-weight" domain objects with no appreciable logic other than their own validation. Models are used to pass data between layers of the stack. In many ways they are similar to ORM objects. Unlike ORM objects, they are NOT themselves allowed to load data from a remote resource. Thought of another way, they cannot import from `conda.gateways`, but rather `conda.gateways` imports from `conda.models` as appropriate to create model objects from remote resources.

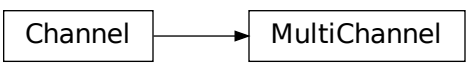
Conda modules importable from `conda.models` are

- `conda.common`
- `conda.models`

### channel

Defines Channel and MultiChannel objects and other channel-related functions.

Object inheritance:



### Classes

| | |
|---|---|
| *ChannelType* | This metaclass does basic caching and enables static constructor method usage with a |
| *Channel* | Channel: |
| *MultiChannel* | Channel: |

### Functions

| | |
|---|---|
| *tokenized_startswith*(→ bool) | |
| *tokenized_conda_url_startswith*(→ bool) | |
| *_get_channel_for_name*(→ Channel) | |
| *_read_channel_configuration*(→ tuple[str \| None, ...) | |
| *parse_conda_channel_url*(→ Channel) | |
| *get_conda_build_local_url*(...) | |
| *prioritize_channels*(→ dict[str, tuple[str, int]]) | |
| *all_channel_urls*(→ boltons.setutils.IndexedSet) | |
| *offline_keep*(→ bool) | |
| *get_channel_objs*(→ tuple[Channel, Ellipsis]) | Return current channels as Channel objects |

**class ChannelType**

Bases: type

This metaclass does basic caching and enables static constructor method usage with a single arg.

**__call__**(**args*, ***kwargs*)

Call self as a function.

**class Channel**(*scheme: str | None = None*, *auth: str | None = None*, *location: str | None = None*, *token: str | None = None*, *name: str | None = None*, *platform: str | None = None*, *package_filename: str | None = None*)

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

**property channel_location: str | None**

**property channel_name: str**

**property subdir: str | None**

**property canonical_name: str**

**property base_url:** `str | None`

**property base_urls:** `tuple[str | None]`

**property subdir_url:** `str`

**property url_channel_wtf:** `tuple[str | None, str]`

**_cache_**

**static _reset_state**() → None

**static from_url**(*url: str*) → *Channel*

**static from_channel_name**(*channel_name: str*) → *Channel*

**static from_value**(*value: str | None*) → *Channel*

Construct a new *Channel* from a single value.

**Parameters**

**value** -- Anyone of the following forms:

***None*, or one of the special strings "<unknown>", "None:///<unknown>", or "None":**
represents the unknown channel, used for packages with unknown origin.

**A URL including a scheme like `file://` or `https://`:**
represents a channel URL.

**A local directory path:**
represents a local channel; relative paths must start with `./`.

**A package file (i.e. the path to a file ending in `.conda` or `.tar.bz2`):**
represents a channel for a single package

**A known channel name:**
represents a known channel, e.g. from the users `.condarc` file or the global configuration.

**Returns**

A channel object.

**static make_simple_channel**(*channel_alias: Channel*, *channel_url: str*, *name: str | None = None*) → *Channel*

**urls**(*with_credentials: bool = False*, *subdirs: collections.abc.Iterable[str] | None = None*) → list[str]

**url**(*with_credentials: bool = False*) → str | None

**__str__**() → str

Return str(self).

**__repr__**() → str

Return repr(self).

**__eq__**(*other: Any*) → bool

Return self==value.

**__hash__**() → int

Return hash(self).

**__nonzero__**() → bool

__bool__() → bool

__json__() → dict[str, Any]

dump() → dict[str, str | None]

class MultiChannel(*name: str*, *channels: collections.abc.Iterable[*Channel*]*, *platform: str | None = None*)

Bases: *Channel*

Channel: scheme <> auth <> location <> token <> channel <> subchannel <> platform <> package_filename

Package Spec: channel <> subchannel <> namespace <> package_name

property channel_location: None

property canonical_name: str

property base_url: None

property base_urls: tuple[str | None, Ellipsis]

urls(*with_credentials: bool = False*, *subdirs: collections.abc.Iterable[str] | None = None*) → list[str]

url(*with_credentials: bool = False*) → None

dump() → dict[str, str | tuple[dict[str, Any], Ellipsis]]

tokenized_startswith(*test_iterable: collections.abc.Iterable[Any]*, *startswith_iterable: collections.abc.Iterable[Any]*) → bool

tokenized_conda_url_startswith(*test_url: collections.abc.Iterable[str]*, *startswith_url: collections.abc.Iterable[str]*) → bool

_get_channel_for_name(*channel_name: str*) → *Channel*

_read_channel_configuration(*scheme: str | None*, *host: str | None*, *port: str | None*, *path: str | None*) → tuple[str | None, str | None, str | None, str | None, str | None]

parse_conda_channel_url(*url: str*) → *Channel*

get_conda_build_local_url() → tuple[conda.common.path.PathType]

prioritize_channels(*channels: collections.abc.Iterable[*Channel | *str]*, *with_credentials: bool = True*, *subdirs: collections.abc.Iterable[str] | None = None*) → dict[str, tuple[str, int]]

all_channel_urls(*channels: collections.abc.Iterable[str | Channel]*, *subdirs: collections.abc.Iterable[str] | None = None*, *with_credentials: bool = True*) → boltons.setutils.IndexedSet

offline_keep(*url: Any*) → bool

get_channel_objs(*ctx:* conda.base.context.Context) → tuple[*Channel*, Ellipsis]

Return current channels as Channel objects

### dist

(Legacy) Low-level implementation of a Channel.

## Classes

| |
|---|
| *DistDetails* |
| *DistType* |
| *Dist* |

## Functions

| |
|---|
| *strip_extension*(original_dist) |
| *split_extension*(original_dist) |
| *dist_str_to_quad*(dist_str) |

**class DistDetails**

Bases: NamedTuple

**name:** str

**version:** str

**build_string:** str

**build_number:** str

**dist_name:** str

**fmt:** str

**class DistType**(*name*, *bases*, *attr*)

Bases: conda.auxlib.entity.EntityType

**__call__**(**args*, ***kwargs*)

Call self as a function.

**strip_extension**(*original_dist*)

**split_extension**(*original_dist*)

**class Dist**(*channel*, *dist_name=None*, *name=None*, *version=None*, *build_string=None*, *build_number=None*, *base_url=None*, *platform=None*, *fmt='.tar.bz2'*)

Bases: *conda.auxlib.entity.Entity*

property full_name

property build

property subdir

property pair

property quad

property is_feature_package

property is_channel

property fn

_cache_

_lazy_validate = True

channel

dist_name

name

fmt

version

build_string

build_number

base_url

platform

to_package_ref()

__str__()
Return str(self).

to_filename(*extension=None*)

to_matchspec()

to_match_spec()

classmethod from_string(*string*, *channel_override=NULL*)

static parse_dist_name(*string*)

classmethod from_url(*url*)

to_url()

__key__()

__lt__(*other*)
Return self<value.

**__gt__**(*other*)
Return self>value.

**__le__**(*other*)
Return self<=value.

**__ge__**(*other*)
Return self>=value.

**__hash__**()
Return hash(self).

**__eq__**(*other*)
Return self==value.

**__ne__**(*other*)
Return self!=value.

**split**(*sep=None*, *maxsplit=-1*)

**rsplit**(*sep=None*, *maxsplit=-1*)

**startswith**(*match*)

**__contains__**(*item*)

**dist_str_to_quad**(*dist_str*)

### enums

Collection of enums used throughout conda.

## Classes

| | |
|---|---|
| *Arch* | Create a collection of name/value pairs. |
| *Platform* | Create a collection of name/value pairs. |
| *FileMode* | Create a collection of name/value pairs. |
| *LinkType* | Create a collection of name/value pairs. |
| *PathType* | Refers to if the file in question is hard linked or soft linked. Originally designed to be used |
| *PackageType* | Create a collection of name/value pairs. |
| *NoarchType* | Create a collection of name/value pairs. |

**class Arch**(**args*, ***kwds*)
Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**x86 = 'x86'**

**x86_64 = 'x86_64'**

**arm64 = 'arm64'**

**armv6l = 'armv6l'**

**armv7l = 'armv7l'**

**aarch64 = 'aarch64'**

**ppc64 = 'ppc64'**

**ppc64le = 'ppc64le'**

**riscv64 = 'riscv64'**

**s390x = 's390x'**

**wasm32 = 'wasm32'**

**z = 'z'**

**classmethod from_sys()**

**__json__()**

**class Platform**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**freebsd = 'freebsd'**

**linux = 'linux'**

**win = 'win32'**

**openbsd = 'openbsd5'**

**osx = 'darwin'**

**zos = 'zos'**

**emscripten = 'emscripten'**

**wasi = 'wasi'**

**classmethod from_sys**()

__json__()

class FileMode(*args, **kwds)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

text = 'text'

binary = 'binary'

__str__()

Return str(self).

class LinkType(*args, **kwds)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**hardlink = 1**

**softlink = 2**

**copy = 3**

**directory = 4**

**__int__**()

**__str__**()

Return str(self).

**__json__**()

**class PathType**(**args*, ***kwds*)

Bases: enum.Enum

Refers to if the file in question is hard linked or soft linked. Originally designed to be used in paths.json

**hardlink = 'hardlink'**

**softlink = 'softlink'**

**directory = 'directory'**

**linked_package_record = 'linked_package_record'**

**pyc_file = 'pyc_file'**

**unix_python_entry_point = 'unix_python_entry_point'**

**windows_python_entry_point_script = 'windows_python_entry_point_script'**

**windows_python_entry_point_exe = 'windows_python_entry_point_exe'**

**basic_types**()

**__str__**()

Return str(self).

**__json__**()

**class PackageType**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

**NOARCH_GENERIC = 'noarch_generic'**

**NOARCH_PYTHON = 'noarch_python'**

**VIRTUAL_PRIVATE_ENV = 'virtual_private_env'**

**VIRTUAL_PYTHON_WHEEL = 'virtual_python_wheel'**

**VIRTUAL_PYTHON_EGG_MANAGEABLE = 'virtual_python_egg_manageable'**

**VIRTUAL_PYTHON_EGG_UNMANAGEABLE = 'virtual_python_egg_unmanageable'**

**VIRTUAL_PYTHON_EGG_LINK = 'virtual_python_egg_link'**

**VIRTUAL_SYSTEM = 'virtual_system'**

**static conda_package_types()**

**static unmanageable_package_types()**

**class NoarchType**(**args*, ***kwds*)

Bases: enum.Enum

Create a collection of name/value pairs.

Example enumeration:

```
>>> class Color(Enum):
...     RED = 1
...     BLUE = 2
...     GREEN = 3
```

Access them by:

- attribute access:

```
>>> Color.RED
<Color.RED: 1>
```

- value lookup:

```
>>> Color(1)
<Color.RED: 1>
```

- name lookup:

```
>>> Color['RED']
<Color.RED: 1>
```

Enumerations can be iterated over, and know how many members they have:

```
>>> len(Color)
3
```

```
>>> list(Color)
[<Color.RED: 1>, <Color.BLUE: 2>, <Color.GREEN: 3>]
```

Methods can be added to enumerations, and members can have their own attributes -- see the documentation for details.

generic = 'generic'

python = 'python'

static coerce(*val*)

### environment

EXPERIMENTAL Conda environment data model

## Classes

| | |
|---|---|
| *EnvironmentConfig* | **Experimental** While experimental, expect both major and minor changes across minor releases. |
| *Environment* | **Experimental** While experimental, expect both major and minor changes across minor releases. |

class EnvironmentConfig

**Experimental** While experimental, expect both major and minor changes across minor releases.

Data model for a conda environment config.

aggressive_update_packages: list[str]

channel_priority: *conda.base.constants.ChannelPriority* | None

channels: list[str]

channel_settings: dict[str, str]

deps_modifier: *conda.base.constants.DepsModifier* | None

disallowed_packages: list[str]

pinned_packages: list[str]

repodata_fns: list[str]

sat_solver: *conda.base.constants.SatSolverChoice* | None

solver: str | None

track_features: list[str]

update_modifier: *conda.base.constants.UpdateModifier* | None

use_only_tar_bz2: bool | None

_append_without_duplicates(*first: list*, *second: list*) → list

_merge(*other:* EnvironmentConfig) → *EnvironmentConfig*

**Experimental** While experimental, expect both major and minor changes across minor releases.

Merges an EnvironmentConfig into this one. Merging rules are: * Primitive types get clobbered if subsequent configs have a value, otherwise keep the last set value * Lists get appended to and deduplicated * Dicts get updated

**classmethod merge**(**configs:* EnvironmentConfig) → *EnvironmentConfig*

**Experimental** While experimental, expect both major and minor changes across minor releases.

Merges a list of EnvironmentConfigs into a single one. Merging rules are: * Primitive types get clobbered if subsequent configs have a value, otherwise keep the last set value * Lists get appended to and deduplicated * Dicts get updated

**class Environment**

**Experimental** While experimental, expect both major and minor changes across minor releases.

Data model for a conda environment.

**prefix:** str

**platform:** str

**config:** *EnvironmentConfig*

**external_packages:** dict[str, list[str]]

**explicit_packages:** list[*conda.models.records.PackageRecord*]

**name:** str | None

**requested_packages:** list[*conda.models.match_spec.MatchSpec*]

**variables:** dict[str, str]

**__post_init__**()

**classmethod merge**(**environments*)

**Experimental** While experimental, expect both major and minor changes across minor releases.

Merges multiple environments into a single environment following the rules: * Keeps first name and/or prefix. * Concatenates and deduplicates requirements. * Reduces configuration and variables (last key wins).

### leased_path_entry

Implements object describing a symbolic link from the base environment to a private environment.

Since private environments are an unrealized feature of conda and has been deprecated this data model no longer serves a purpose and has also been deprecated.

### match_spec

Implements the query language for conda packages (a.k.a, MatchSpec).

The MatchSpec is the conda package specification (e.g. *conda==23.3*, *python<3.7*, *cryptography * *_0*) and is used to communicate the desired packages to install.

### Classes

| | |
|---|---|
| *MatchSpecType* | |
| *MatchSpec* | The query language for conda packages. |
| *MatchInterface* | |
| *_StrMatchMixin* | |
| *ExactStrMatch* | |
| *ExactLowerStrMatch* | |
| *GlobStrMatch* | |
| *GlobLowerStrMatch* | |
| *SplitStrMatch* | |
| *FeatureMatch* | |
| *ChannelMatch* | |
| *CaseInsensitiveStrMatch* | |

### Functions

| | |
|---|---|
| *_parse_version_plus_build*(v_plus_b) | This should reliably pull the build string out of a version + build string combo. |
| *_parse_legacy_dist*(dist_str) | |
| *_parse_channel*(channel_val) | |
| *_parse_spec_str*(spec_str) | |

### Attributes

| | |
|---|---|
| *_PARSE_CACHE* | |
| *_implementors* | |

**class MatchSpecType**

Bases: type

**__call__**(*spec_arg=None*, ***kwargs*)

Call self as a function.

**class MatchSpec**(*optional=False*, *target=None*, ***kwargs*)

The query language for conda packages.

Any of the fields that comprise a PackageRecord can be used to compose a *MatchSpec*.

*MatchSpec* can be composed with keyword arguments, where keys are any of the attributes of PackageRecord. Values for keyword arguments are the exact values the attribute should match against. Many fields can also be matched against non-exact values--by including wildcard * and >/< ranges--where supported. Any non-specified field is the equivalent of a full wildcard match.

*MatchSpec* can also be composed using a single positional argument, with optional keyword arguments. Keyword arguments also override any conflicting information provided in the positional argument. The positional argument can be either an existing *MatchSpec* instance or a string. Conda has historically supported more than one string representation for equivalent *MatchSpec* queries. This *MatchSpec* should accept any existing valid spec string, and correctly compose a *MatchSpec* instance.

A series of rules are now followed for creating the canonical string representation of a *MatchSpec* instance. The canonical string representation can generically be represented by

(channel(/subdir):(namespace):)name(version(build))[key1=value1,key2=value2]

where *()* indicate optional fields. The rules for constructing a canonical string representation are:

1. *name* (i.e. "package name") is required, but its value can be '*'. Its position is always outside the key-value brackets.
2. If *version* is an exact version, it goes outside the key-value brackets and is prepended by ==. If *version* is a "fuzzy" value (e.g. *1.11.**), it goes outside the key-value brackets with the .* left off and is prepended by =. Otherwise *version* is included inside key-value brackets.
3. If *version* is an exact version, and *build* is an exact value, *build* goes outside key-value brackets prepended by a =. Otherwise, *build* goes inside key-value brackets. *build_string* is an alias for *build*.
4. The *namespace* position is being held for a future conda feature.
5. If *channel* is included and is an exact value, a *::* separator is ued between *channel* and *name*. *channel* can either be a canonical channel name or a channel url. In the canonical string representation, the canonical channel name will always be used.
6. If *channel* is an exact value and *subdir* is an exact value, *subdir* is appended to *channel* with a */* separator. Otherwise, *subdir* is included in the key-value brackets.
7. Key-value brackets can be delimited by comma, space, or comma+space. Value can optionally be wrapped in single or double quotes, but must be wrapped if *value* contains a comma, space, or equal sign. The canonical format uses comma delimiters and single quotes.
8. When constructing a *MatchSpec* instance from a string, any key-value pair given inside the key-value brackets overrides any matching parameter given outside the brackets.

When *MatchSpec* attribute values are simple strings, the are interpreted using the following conventions:

- If the string begins with ^ and ends with *$*, it is converted to a regex.
- If the string contains an asterisk (*), it is transformed from a glob to a regex.
- Otherwise, an exact match to the string is sought.

### Examples

```
>>> str(MatchSpec(name='foo', build='py2*', channel='conda-forge'))
'conda-forge::foo[build=py2*]'
>>> str(MatchSpec('foo 1.0 py27_0'))
'foo==1.0=py27_0'
>>> str(MatchSpec('foo=1.0=py27_0'))
'foo==1.0=py27_0'
>>> str(MatchSpec('conda-forge::foo[version=1.0.*]'))
'conda-forge::foo=1.0'
>>> str(MatchSpec('conda-forge/linux-64::foo>=1.0'))
"conda-forge/linux-64::foo[version='>=1.0']"
>>> str(MatchSpec('*/linux-64::foo>=1.0'))
"foo[subdir=linux-64,version='>=1.0']"
```

**To fully-specify a package with a full, exact spec, the fields**

- channel
- subdir
- name
- version
- build

must be given as exact values. In the future, the namespace field will be added to this list. Alternatively, an exact spec is given by '*[md5=12345678901234567890123456789012]' or '*[sha256=f453db4ffe2271ec492a2913af4e61d4a6c118201f07de757df0eff769b65d2e]'.

**property is_name_only_spec**

**property optional**

**property target**

**property original_spec_str**

**property name**

**property strictness**

**property spec**

**property version**

**property fn**

**FIELD_NAMES = ('channel', 'subdir', 'name', 'version', 'build', 'build_number', 'track_features', 'features',...**

**FIELD_NAMES_SET**

**_MATCHER_CACHE**

**classmethod from_dist_str**(*dist_str*)

**get_exact_value**(*field_name*)

**get_raw_value**(*field_name*)

**get**(*field_name*, *default=None*)

**dist_str**()

**match**(*rec*)

Accepts a *PackageRecord* or a dict, and matches can pull from any field in that record. Returns True for a match, and False for no match.

**_match_individual**(*record*, *field_name*, *match_component*)

**_is_simple**()

**_is_single**()

**_to_filename_do_not_use**()

**__repr__**()

Return repr(self).

**__str__**()

Return str(self).

**__json__**()

**conda_build_form**()

**__eq__**(*other*)

Return self==value.

**__hash__**()

Return hash(self).

**_hash_key**()

**__contains__**(*field*)

**_build_components**(***kwargs*)

**static _make_component**(*field_name*, *value*)

**classmethod merge**(*match_specs*, *union=False*)

**classmethod union**(*match_specs*)

**_merge**(*other*, *union=False*)

**_parse_version_plus_build**(*v_plus_b*)

This should reliably pull the build string out of a version + build string combo. .. rubric:: Examples

```
>>> _parse_version_plus_build("=1.2.3 0")
('=1.2.3', '0')
>>> _parse_version_plus_build("1.2.3=0")
('1.2.3', '0')
>>> _parse_version_plus_build(">=1.0 , < 2.0 py34_0")
('>=1.0,<2.0', 'py34_0')
```

(continues on next page)

(continued from previous page)

```
>>> _parse_version_plus_build(">=1.0 , < 2.0 =py34_0")
('>=1.0,<2.0', 'py34_0')
>>> _parse_version_plus_build("=1.2.3 ")
('=1.2.3', None)
>>> _parse_version_plus_build(">1.8,<2|==1.7")
('>1.8,<2|==1.7', None)
>>> _parse_version_plus_build("* openblas_0")
('*', 'openblas_0')
>>> _parse_version_plus_build("* *")
('*', '*')
```

**_parse_legacy_dist**(*dist_str*)

### Examples

```
>>> _parse_legacy_dist("_license-1.1-py27_1.tar.bz2")
('_license', '1.1', 'py27_1')
>>> _parse_legacy_dist("_license-1.1-py27_1")
('_license', '1.1', 'py27_1')
```

**_parse_channel**(*channel_val*)

**_PARSE_CACHE**

**_parse_spec_str**(*spec_str*)

**class MatchInterface**(*value*)

**property raw_value**

**abstract property exact_value**

If the match value is an exact specification, returns the value. Otherwise returns None.

**abstract match**(*other*)

**matches**(*value*)

**merge**(*other*)

**union**(*other*)

**class _StrMatchMixin**

**property exact_value**

**__str__**()

Return str(self).

**__repr__**()

Return repr(self).

**__eq__**(*other*)

Return self==value.

__hash__()
Return hash(self).

class ExactStrMatch(*value*)
Bases: _StrMatchMixin, MatchInterface

__slots__ = ('_raw_value',)

match(*other*)

class ExactLowerStrMatch(*value*)
Bases: ExactStrMatch

match(*other*)

class GlobStrMatch(*value*)
Bases: _StrMatchMixin, MatchInterface

property exact_value
If the match value is an exact specification, returns the value. Otherwise returns None.

property matches_all

__slots__ = ('_raw_value', '_re_match')

match(*other*)

merge(*other*)

class GlobLowerStrMatch(*value*)
Bases: GlobStrMatch

class SplitStrMatch(*value*)
Bases: MatchInterface

property exact_value
If the match value is an exact specification, returns the value. Otherwise returns None.

__slots__ = ('_raw_value',)

_convert(*value*)

match(*other*)

__repr__()
Return repr(self).

__str__()
Return str(self).

__eq__(*other*)
Return self==value.

__hash__()
Return hash(self).

class FeatureMatch(*value*)
Bases: MatchInterface

property exact_value
If the match value is an exact specification, returns the value. Otherwise returns None.

__slots__ = ('_raw_value',)

_convert(*value*)

match(*other*)

__repr__()
Return repr(self).

__str__()
Return str(self).

__eq__(*other*)
Return self==value.

__hash__()
Return hash(self).

class ChannelMatch(*value*)
Bases: *GlobStrMatch*

match(*other*)

__str__()
Return str(self).

__repr__()
Return repr(self).

class CaseInsensitiveStrMatch(*value*)
Bases: *GlobLowerStrMatch*

match(*other*)

_implementors

### package_info

(Legacy) Low-level implementation of a PackageRecord.

## Classes

| | |
|---|---|
| *NoarchField* | Fields are doing something very similar to boxing and unboxing |
| *Noarch* | |
| *PreferredEnv* | |
| *PackageMetadata* | |
| *PackageInfo* | |

class NoarchField(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

box(*instance*, *instance_type*, *val*)

class Noarch(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

type

entry_points

class PreferredEnv(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

name

executable_paths

softlink_paths

class PackageMetadata(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

package_metadata_version

noarch

preferred_env

class PackageInfo(***kwargs*)

Bases: *conda.auxlib.entity.ImmutableEntity*

property name

property version

property build

property build_number

extracted_package_dir

**package_tarball_full_path**

**channel**

**repodata_record**

**url**

**icondata**

**package_metadata**

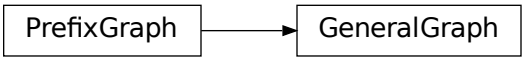
**paths_data**

**dist_str**()

### prefix_graph

Implements directed graphs to sort and manipulate packages within a prefix.

Object inheritance:




## Classes

| | |
|---|---|
| *PrefixGraph* | A directed graph structure used for sorting packages (prefix_records) in prefixes and |
| *GeneralGraph* | Compared with PrefixGraph, this class takes in more than one record of a given name, |

**class PrefixGraph**(*records*, *specs=()*)

A directed graph structure used for sorting packages (prefix_records) in prefixes and manipulating packages within prefixes (e.g. removing and pruning).

The terminology used for edge direction is "parents" and "children" rather than "successors" and "predecessors". The parent nodes of a record are those records in the graph that match the record's "depends" field. E.g. NodeA depends on NodeB, then NodeA is a child of NodeB, and NodeB is a parent of NodeA. Nodes can have zero parents, or more than two parents.

Most public methods mutate the graph.

**property records**

**remove_spec**(*spec*)

Remove all matching nodes, and any associated child nodes.

**Parameters**
**spec** (MatchSpec)

**Returns**
The removed nodes.

**Return type**
tuple[*PrefixRecord*]

**remove_youngest_descendant_nodes_with_specs**()

A specialized method used to determine only dependencies of requested specs.

**Returns**
The removed nodes.

**Return type**
tuple[*PrefixRecord*]

**prune**()

Prune back all packages until all child nodes are anchored by a spec.

**Returns**
The pruned nodes.

**Return type**
tuple[*PrefixRecord*]

**get_node_by_name**(*name*)

**all_descendants**(*node*)

**all_ancestors**(*node*)

**_remove_node**(*node*)

Removes this node and all edges referencing it.

**_toposort**()

**classmethod _toposort_raise_on_cycles**(*graph*)

**classmethod _topo_sort_handle_cycles**(*graph*)

**static _toposort_pop_key**(*graph*)

Pop an item from the graph that has the fewest parents. In the case of a tie, use the node with the alphabetically-first package name.

**static _toposort_prepare_graph**(*graph*)

**class GeneralGraph**(*records*, *specs=()*)

Bases: *PrefixGraph*

Compared with PrefixGraph, this class takes in more than one record of a given name, and operates on that graph from the higher view across any matching dependencies. It is not a Prefix thing, but more like a "graph of all possible candidates" thing, and is used for unsatisfiability analysis

**breadth_first_search_by_name**(*root_spec*, *target_spec*)

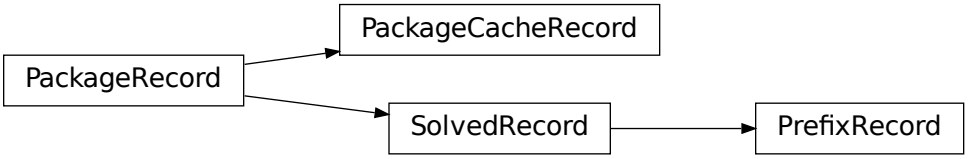
Return shorted path from root_spec to spec_name

### records

Implements the data model for conda packages.

A PackageRecord is the record of a package present in a channel. A PackageCache is the record of a downloaded and cached package. A PrefixRecord is the record of a package installed into a conda environment.

Object inheritance:



### Classes

| | |
|---|---|
| *LinkTypeField* | Fields are doing something very similar to boxing and unboxing |
| *NoarchField* | Fields are doing something very similar to boxing and unboxing |
| *TimestampField* | Fields are doing something very similar to boxing and unboxing |
| *Link* | |
| *_FeaturesField* | Fields are doing something very similar to boxing and unboxing |
| *ChannelField* | Fields are doing something very similar to boxing and unboxing |
| *SubdirField* | Fields are doing something very similar to boxing and unboxing |
| *FilenameField* | Fields are doing something very similar to boxing and unboxing |
| *PackageTypeField* | Fields are doing something very similar to boxing and unboxing |
| *PathData* | |
| *PathDataV1* | |
| *PathsData* | |
| *PackageRecord* | Representation of a concrete package archive (tarball or .conda file). |
| *Md5Field* | Fields are doing something very similar to boxing and unboxing |
| *PackageCacheRecord* | Representation of a package that has been downloaded or unpacked in the local package cache. |
| *SolvedRecord* | Representation of a package that has been returned as part of a solver solution. |
| *PrefixRecord* | Representation of a package that is installed in a local conda environmnet. |

### Attributes

| |
|---|
| *EMPTY_LINK* |

**class LinkTypeField**(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**box**(*instance*, *instance_type*, *val*)

**class NoarchField**(*enum_class*, *default=NULL*, *required=True*, *validation=None*, *in_dump=True*, *default_in_dump=True*, *nullable=False*, *immutable=False*, *aliases=()*)

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**box**(*instance*, *instance_type*, *val*)

**class TimestampField**

Bases: *conda.auxlib.entity.NumberField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**static _make_seconds**(*val*)

**static _make_milliseconds**(*val*)

**box**(*instance*, *instance_type*, *val*)

**dump**(*instance*, *instance_type*, *val*)

__get__(*instance*, *instance_type*)

class Link(***kwargs*)

Bases: conda.auxlib.entity.DictSafeMixin, *conda.auxlib.entity.Entity*

source

type

EMPTY_LINK

class _FeaturesField(***kwargs*)

Bases: *conda.auxlib.entity.ListField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

box(*instance*, *instance_type*, *val*)

dump(*instance*, *instance_type*, *val*)

class ChannelField(*aliases=()*)

Bases: *conda.auxlib.entity.ComposableField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

dump(*instance*, *instance_type*, *val*)

__get__(*instance*, *instance_type*)

class SubdirField

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**__get__**(*instance*, *instance_type*)

**class FilenameField**(*aliases=()*)

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**__get__**(*instance*, *instance_type*)

**class PackageTypeField**

Bases: *conda.auxlib.entity.EnumField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.
- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**__get__**(*instance*, *instance_type*)

**class PathData**(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

**property path**

**_path**

**prefix_placeholder**

**file_mode**

**no_link**

**path_type**

**class PathDataV1**(***kwargs*)

Bases: *PathData*

**sha256**

**size_in_bytes**

**inode_paths**

**sha256_in_prefix**

**class PathsData**(***kwargs*)

Bases: *conda.auxlib.entity.Entity*

**paths_version**

**paths**

**class PackageRecord**(**args*, ***kwargs*)

Bases: conda.auxlib.entity.DictSafeMixin, *conda.auxlib.entity.Entity*

Representation of a concrete package archive (tarball or .conda file).

It captures all the relevant information about a given package archive, including its source, in the following attributes.

Note that there are three subclasses, *SolvedRecord*, *PrefixRecord* and *PackageCacheRecord*. These capture the same information, but are augmented with additional information relevant for these sources of packages.

Further note that *PackageRecord* makes use of its *_pkey* for comparison and hash generation. This means that for common operations, like comparisons between *PackageRecord* s and reference of *PackageRecord* s in mappings, _different_ objects appear identical. The fields taken into account are marked in the following list of attributes. The subclasses do not add further attributes to the *_pkey*.

**property channel_name: str | None**

The canonical name of the channel of this package.

Part of the *_pkey*.

**Type**
str

**property schannel**

**property _pkey**

The components of the PackageRecord that are used for comparison and hashing.

The *_pkey* is a tuple made up of the following fields of the *PackageRecord*. Two *PackageRecord* s test equal if their respective *_pkey* s are equal. The hash of the *PackageRecord* (important for dictionary access) is the hash of the *_pkey*.

The included fields are:

- *channel_name*

- *subdir*
- *name*
- *version*
- *build_number*
- *build*
- *fn* only if *separate_format_cache* is set to true (default: false)

**Type**
tuple

property is_unmanageable

property combined_depends

property namekey

name

version

build

build_number

channel

subdir

fn

md5

legacy_bz2_md5

legacy_bz2_size

url

sha256

arch

platform

depends

constrains

track_features

features

noarch

preferred_env

python_site_packages_path

**license**

**license_family**

**package_type**

**timestamp**

**date**

**size**

**metadata:** set[str]

**__hash__**()
Return hash(self).

**__eq__**(*other*)
Return self==value.

**dist_str**(*canonical_name: bool = True*) → str

**dist_fields_dump**()

**__str__**()
Return str(self).

**to_match_spec**()

**to_simple_match_spec**()

**record_id**()

**classmethod feature**(*feature_name*) → *PackageRecord*

**classmethod virtual_package**(*name: str*, *version: str | None = None*, *build_string: str | None = None*) → *PackageRecord*

Create a virtual package record.

**Parameters**

- **name** -- The name of the virtual package.
- **version** -- The version of the virtual package, defaults to "0".
- **build_string** -- The build string of the virtual package, defaults to "0".

**Returns**
A PackageRecord representing the virtual package.

**class Md5Field**

Bases: *conda.auxlib.entity.StringField*

Fields are doing something very similar to boxing and unboxing of c#/java primitives. __set__ should take a "primitive" or "raw" value and create a "boxed" or "programmatically usable" value of it. While __get__ should return the boxed value, dump in turn should unbox the value into a primitive or raw value.

**Parameters**

- **types** (*primitive literal or type or sequence of types*)
- **default** (*any, callable, optional*) -- If default is callable, it's guaranteed to return a valid value at the time of Entity creation.

- **required** (*boolean, optional*)
- **validation** (*callable, optional*)
- **dump** (*boolean, optional*)

**__get__**(*instance*, *instance_type*)

**class PackageCacheRecord**(**args*, ***kwargs*)

Bases: *PackageRecord*

Representation of a package that has been downloaded or unpacked in the local package cache.

Specialization of *PackageRecord* that adds information for packages that exist in the local package cache, either as the downloaded package file, or unpacked in its own package dir, or both.

Note that this class does not add new fields to the *PackageRecord._pkey* so that a pure *PackageRecord* object that has the same _pkey fields as a different *PackageCacheRecord* object (or, indeed, a *PrefixRecord* object) will be considered equal and will produce the same hash.

**property is_fetched**

Whether the package file exists locally.

**Type**
bool

**property is_extracted**

Whether the package has been extracted locally.

**Type**
bool

**property tarball_basename**

The basename of the local package file.

**Type**
str

**package_tarball_full_path**

**extracted_package_dir**

**md5**

**_calculate_md5sum**()

**class SolvedRecord**(**args*, ***kwargs*)

Bases: *PackageRecord*

Representation of a package that has been returned as part of a solver solution.

This sits between *PackageRecord* and *PrefixRecord*, simply adding `requested_spec` so it can be used in lockfiles without requiring the artifact on disk.

**requested_spec**

**class PrefixRecord**(**args*, ***kwargs*)

Bases: *SolvedRecord*

Representation of a package that is installed in a local conda environmnet.

Specialization of *PackageRecord* that adds information for packages that are installed in a local conda environment or prefix.

Note that this class does not add new fields to the *PackageRecord._pkey* so that a pure *PackageRecord* object that has the same _pkey fields as a different *PrefixRecord* object (or, indeed, a *PackageCacheRecord* object) will be considered equal and will produce the same hash.

Objects of this class are generally constructed from metadata in json files inside *$prefix/conda-meta*.

**package_tarball_full_path**

**extracted_package_dir**

**files**

**paths_data**

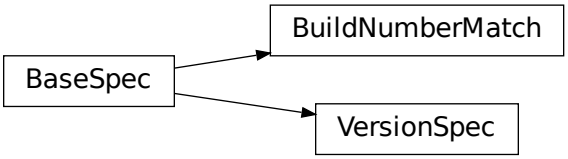
**link**

**auth**

### version

Implements the version spec with parsing and comparison logic.

Object inheritance:




## Classes

| | |
|---|---|
| *SingleStrArgCachingType* | |
| *VersionOrder* | Implement an order relation between version strings. |
| *BaseSpec* | |
| *VersionSpec* | |
| *BuildNumberMatch* | |

### Functions

| | |
|---|---|
| *normalized_version*(→ VersionOrder) | Parse a version string and return VersionOrder object. |
| *ver_eval*(vtest, spec) | |
| *treeify*(spec_str) | |
| *untreeify*(spec[, _inand, depth]) | |
| *compatible_release_operator*(x, y) | |

### Attributes

| |
|---|
| *version_check_re* |
| *version_split_re* |
| *version_cache* |
| *VSPEC_TOKENS* |
| *version_relation_re* |
| *regex_split_re* |
| *OPERATOR_MAP* |
| *OPERATOR_START* |
| *VersionMatch* |

**normalized_version**(*version: str*) → *VersionOrder*

Parse a version string and return VersionOrder object.

**ver_eval**(*vtest*, *spec*)

**version_check_re**

**version_split_re**

**version_cache**

**class SingleStrArgCachingType**

Bases: `type`

**__call__**(*arg*)

Call self as a function.

class VersionOrder(*vstr: str*)

Implement an order relation between version strings.

Version strings can contain the usual alphanumeric characters (A-Za-z0-9), separated into components by dots and underscores. Empty segments (i.e. two consecutive dots, a leading/trailing underscore) are not permitted. An optional epoch number - an integer followed by '!' - can proceed the actual version string (this is useful to indicate a change in the versioning scheme itself). Version comparison is case-insensitive.

Conda supports six types of version strings: * Release versions contain only integers, e.g. '1.0', '2.3.5'. * Pre-release versions use additional letters such as 'a' or 'rc',

for example '1.0a1', '1.2.beta3', '2.3.5rc3'.

- Development versions are indicated by the string 'dev', for example '1.0dev42', '2.3.5.dev12'.
- Post-release versions are indicated by the string 'post', for example '1.0post1', '2.3.5.post2'.
- Tagged versions have a suffix that specifies a particular property of interest, e.g. '1.1.parallel'. Tags can be added to any of the preceding four types. As far as sorting is concerned, tags are treated like strings in pre-release versions.
- An optional local version string separated by '+' can be appended to the main (upstream) version string. It is only considered in comparisons when the main versions are equal, but otherwise handled in exactly the same manner.

To obtain a predictable version ordering, it is crucial to keep the version number scheme of a given package consistent over time. Specifically, * version strings should always have the same number of components

(except for an optional tag suffix or local version string),

- letters/strings indicating non-release versions should always occur at the same position.

Before comparison, version strings are parsed as follows: * They are first split into epoch, version number, and local version

number at '!' and '+' respectively. If there is no '!', the epoch is set to 0. If there is no '+', the local version is empty.

- The version part is then split into components at '.' and '_'.
- Each component is split again into runs of numerals and non-numerals
- Subcomponents containing only numerals are converted to integers.
- Strings are converted to lower case, with special treatment for 'dev' and 'post'.
- When a component starts with a letter, the fillvalue 0 is inserted to keep numbers and strings in phase, resulting in '1.1.a1' == 1.1.0a1'.
- The same is repeated for the local version part.

### Examples

1.2g.beta15.rc => [[0], [1], [2, 'g'], [0, 'beta', 15], [0, 'rc']] 1!2.15.1_ALPHA => [[1], [2], [15], [1, '_alpha']]

The resulting lists are compared lexicographically, where the following rules are applied to each pair of corresponding subcomponents: * integers are compared numerically * strings are compared lexicographically, case-insensitive * strings are smaller than integers, except * 'dev' versions are smaller than all corresponding versions of other types * 'post' versions are greater than all corresponding versions of other types * if a subcomponent has no correspondent, the missing correspondent is

treated as integer 0 to ensure '1.1' == '1.1.0'.

**The resulting order is:**

0.4

< 0.4.0 < 0.4.1.rc

**== 0.4.1.RC # case-insensitive comparison**
< 0.4.1 < 0.5a1 < 0.5b3 < 0.5C1 # case-insensitive comparison < 0.5 < 0.9.6 < 0.960923 < 1.0 < 1.1dev1 # special case 'dev' < 1.1_ # appended underscore is special case for openssl-like versions < 1.1a1 < 1.1.0dev1 # special case 'dev'

**== 1.1.dev1 # 0 is inserted before string**
< 1.1.a1 < 1.1.0rc1 < 1.1.0

**== 1.1**
< 1.1.0post1 # special case 'post'

**== 1.1.post1 # 0 is inserted before string**
< 1.1post1 # special case 'post' < 1996.07.12 < 1!0.4.1 # epoch increased < 1!3.1.1.6 < 2!0.4.1 # epoch increased again

Some packages (most notably openssl) have incompatible version conventions. In particular, openssl interprets letters as version counters rather than pre-release identifiers. For openssl, the relation

1.0.1 < 1.0.1a => False # should be true for openssl

holds, whereas conda packages use the opposite ordering. You can work-around this problem by appending an underscore to plain version numbers:

1.0.1_ < 1.0.1a => True # ensure correct ordering for openssl

**_cache_**

**__str__**() → str
Return str(self).

**__repr__**() → str
Return repr(self).

**_eq**(*t1: list[str]*, *t2: list[str]*) → bool

**__eq__**(*other: object*) → bool
Return self==value.

**startswith**(*other: object*) → bool

**__ne__**(*other: object*) → bool
Return self!=value.

**__lt__**(*other: object*) → bool

Return self<value.

**__gt__**(*other: object*) → bool

Return self>value.

**__le__**(*other: object*) → bool

Return self<=value.

**__ge__**(*other: object*) → bool

Return self>=value.

**VSPEC_TOKENS** = '\\s*\\^[^$]*[$]|\\s*[()|,]|[^()|,]+'

**treeify**(*spec_str*)

### Examples

```
>>> treeify("1.2.3")
'1.2.3'
>>> treeify("1.2.3,>4.5.6")
(',', '1.2.3', '>4.5.6')
>>> treeify("1.2.3,4.5.6|<=7.8.9")
('|', (',', '1.2.3', '4.5.6'), '<=7.8.9')
>>> treeify("(1.2.3|4.5.6),<=7.8.9")
(',', ('|', '1.2.3', '4.5.6'), '<=7.8.9')
>>> treeify("((1.5|((1.6|1.7), 1.8), 1.9 |2.0))|2.1")
('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1')
>>> treeify("1.5|(1.6|1.7),1.8,1.9|2.0|2.1")
('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1')
```

**untreeify**(*spec*, *_inand=False*, *depth=0*)

### Examples

```
>>> untreeify('1.2.3')
'1.2.3'
>>> untreeify((',', '1.2.3', '>4.5.6'))
'1.2.3,>4.5.6'
>>> untreeify(('|', (',', '1.2.3', '4.5.6'), '<=7.8.9'))
'(1.2.3,4.5.6)|<=7.8.9'
>>> untreeify((',', ('|', '1.2.3', '4.5.6'), '<=7.8.9'))
'(1.2.3|4.5.6),<=7.8.9'
>>> untreeify(('|', '1.5', (',', ('|', '1.6', '1.7'), '1.8', '1.9'), '2.0', '2.1'))
'1.5|((1.6|1.7),1.8,1.9)|2.0|2.1'
```

**compatible_release_operator**(*x*, *y*)

**version_relation_re**

**regex_split_re**

**OPERATOR_MAP**

OPERATOR_START

class BaseSpec(*spec_str*, *matcher*, *is_exact*)

property spec

property raw_value

property exact_value

is_exact()

__eq__(*other*)
Return self==value.

__ne__(*other*)
Return self!=value.

__hash__()
Return hash(self).

__str__()
Return str(self).

__repr__()
Return repr(self).

abstract merge(*other*)

regex_match(*spec_str*)

operator_match(*spec_str*)

any_match(*spec_str*)

all_match(*spec_str*)

exact_match(*spec_str*)

always_true_match(*spec_str*)

class VersionSpec(*vspec*)
Bases: *BaseSpec*

_cache_

get_matcher(*vspec*)

merge(*other*)

union(*other*)

VersionMatch

class BuildNumberMatch(*vspec*)
Bases: *BaseSpec*

property exact_value: int | None

_cache_

**get_matcher**(*vspec*)

**merge**(*other*)

**union**(*other*)

**__str__**()
Return str(self).

**__repr__**()
Return repr(self).

### notices

### cache

**Handles all caching logic including:**

- Retrieving from cache
- Saving to cache
- Determining whether not certain items have expired and need to be refreshed

## Functions

| | |
|---|---|
| *cached_response*(func) | |
| *is_notice_response_cache_expired*(→ bool) | This checks the contents of the cache response to see if it is expired. |
| *get_notices_cache_dir*(→ pathlib.Path) | Returns the location of the notices cache directory as a Path object |
| *get_notices_cache_file*(→ pathlib.Path) | Return path of notices cache |
| *get_notice_response_from_cache*(...) | Retrieves a notice response object from cache if it exists. |
| *write_notice_response_to_cache*(→ None) | Writes our notice data to our local cache location. |
| *mark_channel_notices_as_viewed*(→ None) | Insert channel notice into our database marking it as read. |
| *get_viewed_channel_notice_ids*(→ set[str]) | Return the ids of the channel notices which have already been seen. |
| *clear_cache*(→ None) | Removes all files in notices cache |

## Attributes

| |
|---|
| *logger* |

**logger**

**cached_response**(*func*)

**is_notice_response_cache_expired**(*channel_notice_response:* conda.notices.types.ChannelNoticeResponse) → bool

This checks the contents of the cache response to see if it is expired.

If for whatever reason we encounter an exception while parsing the individual messages, we assume an invalid cache and return true.

**get_notices_cache_dir**() → pathlib.Path

Returns the location of the notices cache directory as a Path object

**get_notices_cache_file**() → pathlib.Path

Return path of notices cache

If the file does not exist, we create it with natural filesystem timestamps, then set only the modification time to be in the past. This ensures notices are checked and displayed immediately rather than waiting for the full display interval.

**get_notice_response_from_cache**(*url: str*, *name: str*, *cache_dir: pathlib.Path*) → *conda.notices.types.ChannelNoticeResponse* | None

Retrieves a notice response object from cache if it exists.

**write_notice_response_to_cache**(*channel_notice_response:* conda.notices.types.ChannelNoticeResponse, *cache_dir: pathlib.Path*) → None

Writes our notice data to our local cache location.

**mark_channel_notices_as_viewed**(*cache_file: pathlib.Path*, *channel_notices:* *collections.abc.Sequence*[conda.notices.types.ChannelNotice]) → None

Insert channel notice into our database marking it as read.

**get_viewed_channel_notice_ids**(*cache_file: pathlib.Path*, *channel_notices:* *collections.abc.Sequence*[conda.notices.types.ChannelNotice]) → set[str]

Return the ids of the channel notices which have already been seen.

**clear_cache**() → None

Removes all files in notices cache

### core

Core conda notices logic.

### Functions

| | |
|---|---|
| *retrieve_notices*(...) | Function used for retrieving notices. This is called by the "notices" decorator as well |
| *display_notices*(→ None) | Prints the channel notices to std out. |
| *notices*(func) | Wrapper for "execute" entry points for subcommands. |
| *get_channel_name_and_urls*(→ list[tuple[ChannelUrl, ...) | Return a sequence of Channel URL and name tuples. |
| *flatten_notice_responses*(...) | |
| *filter_notices*(...) | Perform filtering actions for the provided sequence of ChannelNotice objects. |
| *is_channel_notices_enabled*(→ bool) | Determines whether channel notices are enabled and therefore displayed when |
| *is_channel_notices_cache_expired*(→ bool) | Checks to see if the notices cache file we use to keep track of |

### Attributes

| |
|---|
| *ChannelName* |
| *ChannelUrl* |
| *logger* |

**ChannelName**

**ChannelUrl**

**logger**

**retrieve_notices**(*limit: int | None = None*, *always_show_viewed: bool = True*, *silent: bool = False*) → *conda.notices.types.ChannelNoticeResultSet*

Function used for retrieving notices. This is called by the "notices" decorator as well as the sub-command "notices"

**Parameters**

- **limit** -- Limit the number of notices to show (defaults to None).
- **always_show_viewed** -- Whether all notices should be shown, not only the unread ones (defaults to True).
- **silent** -- Whether to use a spinner when fetching and caching notices.

**display_notices**(*channel_notice_set:* conda.notices.types.ChannelNoticeResultSet) → None

Prints the channel notices to std out.

**notices**(*func*)

Wrapper for "execute" entry points for subcommands.

If channel notices need to be fetched, we do that first and then run the command normally. We then display these notices at the very end of the command output so that the user is more likely to see them.

**This ordering was specifically done to address the following bug report:**

- https://github.com/conda/conda/issues/11847

**Parameters**
**func** -- Function to be decorated

**get_channel_name_and_urls**(*channels: collections.abc.Sequence[conda.models.channel.Channel | conda.models.channel.MultiChannel]*) → list[tuple[ChannelUrl, ChannelName]]

Return a sequence of Channel URL and name tuples.

This function handles both Channel and MultiChannel object types.

**flatten_notice_responses**(*channel_notice_responses: collections.abc.Sequence[conda.notices.types.ChannelNoticeResponse]*) → collections.abc.Sequence[*conda.notices.types.ChannelNotice*]

**filter_notices**(*channel_notices: collections.abc.Sequence[conda.notices.types.ChannelNotice]*, *limit: int | None = None*, *exclude: set[str] | None = None*) → collections.abc.Sequence[*conda.notices.types.ChannelNotice*]

Perform filtering actions for the provided sequence of ChannelNotice objects.

**is_channel_notices_enabled**(*ctx:* conda.base.context.Context) → bool

Determines whether channel notices are enabled and therefore displayed when invoking the *notices* command decorator.

**This only happens when:**

- offline is False
- number_channel_notices is greater than 0

**Parameters**
**ctx** -- The conda context object

**is_channel_notices_cache_expired**() → bool

Checks to see if the notices cache file we use to keep track of displayed notices is expired. This involves checking the mtime attribute of the file. Anything older than what is specified as the NOTICES_DECORATOR_DISPLAY_INTERVAL is considered expired.

### fetch

Notices network fetch logic.

## Functions

| | |
|---|---|
| *get_notice_responses*(...) | Provided a list of channel notification url/name tuples, return a sequence of |
| *get_channel_notice_response*(...) | Return a channel response object. We use this to wrap the response with |

### Attributes

| |
|---|
| *logger* |

**logger**

**get_notice_responses**(*url_and_names: collections.abc.Sequence[tuple[str, str]]*, *silent: bool = False*, *max_workers: int = 10*) → collections.abc.Sequence[*conda.notices.types.ChannelNoticeResponse*]

Provided a list of channel notification url/name tuples, return a sequence of ChannelNoticeResponse objects.

**Parameters**

- **url_and_names** -- channel url and the channel name
- **silent** -- turn off "loading animation" (defaults to False)
- **max_workers** -- increase worker number in thread executor (defaults to 10)

**Returns**

Sequence[ChannelNoticeResponse]

**get_channel_notice_response**(*url: str*, *name: str*) → *conda.notices.types.ChannelNoticeResponse* | None

Return a channel response object. We use this to wrap the response with additional channel information to use. If the response was invalid we suppress/log and error message.

## types

Implements all conda.notices types.

### Classes

| | |
|---|---|
| *ChannelNotice* | Represents an individual channel notice. |
| *ChannelNoticeResultSet* | Represents a list of a channel notices, plus some accompanying |
| *ChannelNoticeResponse* | |

### Attributes

| |
|---|
| *UNDEFINED_MESSAGE_ID* |

**UNDEFINED_MESSAGE_ID = 'undefined'**

class ChannelNotice

Bases: NamedTuple

Represents an individual channel notice.

id: str

channel_name: str | None

message: str | None

level: *conda.base.constants.NoticeLevel*

created_at: datetime.datetime | None

expired_at: datetime.datetime | None

interval: int | None

to_dict()

class ChannelNoticeResultSet

Bases: NamedTuple

Represents a list of a channel notices, plus some accompanying metadata such as *viewed_channel_notices*.

channel_notices: collections.abc.Sequence[*ChannelNotice*]

total_number_channel_notices: int

viewed_channel_notices: int

class ChannelNoticeResponse

Bases: NamedTuple

property notices: collections.abc.Sequence[*ChannelNotice*]

url: str

name: str

json_data: dict | None

static _parse_notice_level(*level: str | None*) → *conda.base.constants.NoticeLevel*

We use this to validate notice levels and provide reasonable defaults if any are invalid.

static _parse_iso_timestamp(*iso_timestamp: str | None*) → datetime.datetime | None

Parse ISO timestamp and fail over to a default value of none.

classmethod get_cache_key(*url: str*, *cache_dir: pathlib.Path*) → pathlib.Path

Returns where this channel response will be cached by hashing the URL.

### views

Handles all display/view logic.

## Functions

| | |
|---|---|
| print_notices(channel_notices) | Accepts a list of channel notice responses and prints a display. |
| print_notice_message(→ None) | Prints a single channel notice. |
| print_more_notices_message(→ None) | Conditionally shows a message informing users how many more message there are. |

**print_notices**(*channel_notices: collections.abc.Sequence[conda.notices.types.ChannelNotice]*)

Accepts a list of channel notice responses and prints a display.

**Parameters**
**channel_notices** -- A sequence of ChannelNotice objects.

**print_notice_message**(*notice:* conda.notices.types.ChannelNotice, *indent: str = ' '*) → None

Prints a single channel notice.

**print_more_notices_message**(*total_notices: int*, *displayed_notices: int*, *viewed_notices: int*) → None

Conditionally shows a message informing users how many more message there are.

## Functions

| | |
|---|---|
| notices(func) | Wrapper for "execute" entry points for subcommands. |

**notices**(*func*)

Wrapper for "execute" entry points for subcommands.

If channel notices need to be fetched, we do that first and then run the command normally. We then display these notices at the very end of the command output so that the user is more likely to see them.

**This ordering was specifically done to address the following bug report:**

- https://github.com/conda/conda/issues/11847

**Parameters**
**func** -- Function to be decorated

### plan

Handle the planning of installs and their execution.

> **Note**
>
> conda.install uses canonical package names in its interface functions, whereas conda.resolve uses package filenames, as those are used as index keys. We try to keep fixes to this "impedance mismatch" local to this module.

### plugins

In this module, you will find everything relevant to conda's plugin system. It contains all of the code that plugin authors will use to write plugins, as well as conda's internal implementations of plugins.

**Modules relevant for plugin authors**

- *conda.plugins.hookspec*: all available hook specifications are listed here, including examples of how to use them
- *conda.plugins.types*: important types to use when defining plugin hooks

**Modules relevant for internal development**

- *conda.plugins.manager*: includes our custom subclass of pluggy's PluginManager class

**Modules with internal plugin implementations**

- *conda.plugins.solvers*: implementation of the "classic" solver
- *conda.plugins.subcommands.doctor*: `conda doctor` subcommand
- *conda.plugins.virtual_packages*: registers virtual packages in conda

### config

**Handles plugin configuration functionality including:**

- Managing plugin-specific settings
- Processing configuration data from various sources
- Dynamically adding and removing plugin settings
- Providing a standardized interface for plugins to access their configurations

## Classes

| | |
|---|---|
| *PluginConfig* | Class used to hold settings for conda plugins. |

**class PluginConfig**(*data*)

Bases: *conda.common.configuration.Configuration*

Class used to hold settings for conda plugins.

The object created by this class should only be accessed via *conda.base.context.Context.plugins*.

When this class is updated via the *add_plugin_setting()* function it adds new setting properties which can be accessed later via the context object.

We currently call that function in *conda.plugins.manager.CondaPluginManager.load_settings()*. because `CondaPluginManager` has access to all registered plugin settings via the settings plugin hook.

**property raw_data: dict[pathlib.Path, dict[str, *conda.common.configuration.RawParameter*]]**

This is used to move everything under the key "plugins" from the provided dictionary to the top level of the returned dictionary. The returned dictionary is then passed to *PluginConfig*.

**property category_map: dict[str, tuple[str, Ellipsis]]**

**parameter_names: tuple[str, Ellipsis] = ()**

**parameter_names_and_aliases: tuple[str, Ellipsis] = ()**

**classmethod add_plugin_setting**(*name: str*, *parameter:* conda.common.configuration.Parameter, *aliases: tuple[str, Ellipsis] = ()*)

Adds a setting to the *PluginConfig* class

**classmethod remove_all_plugin_settings**() → None

Removes all attached settings from the *PluginConfig* class

**get_descriptions**() → dict[str, str]

**describe_parameter**(*parameter_name*) → dict[str, Any]

Returns the description of a parameter.

We add to this method in order to change the "name" key that is returned to prepend "plugins." to it.

### environment_specifiers

Register the built-in environment specifier hook implementations.

### binstar

**EXPERIMENTAL** Register the conda env spec for binstar specs.

## Functions

*conda_environment_specifiers*()

**conda_environment_specifiers**()

### environment_yml

**EXPERIMENTAL** Register the conda env spec for environment.yml files.

#### Functions

*conda_environment_specifiers*()

**conda_environment_specifiers**()

### explicit

**EXPERIMENTAL** Register the conda env spec for explicit files.

#### Functions

*conda_environment_specifiers*()

**conda_environment_specifiers**()

### requirements_txt

**EXPERIMENTAL** Register the conda env spec for requirements.txt files.

#### Functions

*conda_environment_specifiers*()

**conda_environment_specifiers**()

**plugins**

### hookspec

Pluggy hook specifications ("hookspecs") to register conda plugins.

Each hookspec defined in *CondaSpecs* contains an example of how to use it.

## Classes

| | |
|---|---|
| *CondaSpecs* | The conda plugin hookspecs, to be used by developers. |

## Attributes

| | |
|---|---|
| *spec_name* | Name used for organizing conda hook specifications |
| *_hookspec* | The conda plugin hook specifications, to be used by developers |
| *hookimpl* | Decorator used to mark plugin hook implementations |

**spec_name = 'conda'**

Name used for organizing conda hook specifications

**_hookspec**

The conda plugin hook specifications, to be used by developers

**hookimpl**

Decorator used to mark plugin hook implementations

**class CondaSpecs**

The conda plugin hookspecs, to be used by developers.

**conda_solvers**() → collections.abc.Iterable[*conda.plugins.types.CondaSolver*]

Register solvers in conda.

**Example:**

```
import logging

from conda import plugins
from conda.core import solve

log = logging.getLogger(__name__)


class VerboseSolver(solve.Solver):
    def solve_final_state(self, *args, **kwargs):
        log.info("My verbose solver!")
        return super().solve_final_state(*args, **kwargs)


@plugins.hookimpl
def conda_solvers():
```

(continues on next page)

(continued from previous page)

```
    yield plugins.CondaSolver(
        name="verbose-classic",
        backend=VerboseSolver,
    )
```

**Returns**
An iterable of solver entries.

**conda_subcommands**() → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

Register external subcommands in conda.

**Example:**

```
from conda import plugins


def example_command(args):
    print("This is an example command!")


@plugins.hookimpl
def conda_subcommands():
    yield plugins.CondaSubcommand(
        name="example",
        summary="example command",
        action=example_command,
    )
```

**Returns**
An iterable of subcommand entries.

**conda_virtual_packages**() → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

Register virtual packages in Conda.

**Example:**

```
from conda import plugins


@plugins.hookimpl
def conda_virtual_packages():
    yield plugins.CondaVirtualPackage(
        name="my_custom_os",
        version="1.2.3",
        build="x86_64",
    )
```

**Returns**
An iterable of virtual package entries.

**conda_pre_commands**() → collections.abc.Iterable[*conda.plugins.types.CondaPreCommand*]

Register pre-command functions in conda.

**Example:**

```
from conda import plugins


def example_pre_command(command):
    print("pre-command action")


@plugins.hookimpl
def conda_pre_commands():
    yield plugins.CondaPreCommand(
        name="example-pre-command",
        action=example_pre_command,
        run_for={"install", "create"},
    )
```

**conda_post_commands**() → collections.abc.Iterable[*conda.plugins.types.CondaPostCommand*]

Register post-command functions in conda.

**Example:**

```
from conda import plugins


def example_post_command(command):
    print("post-command action")


@plugins.hookimpl
def conda_post_commands():
    yield plugins.CondaPostCommand(
        name="example-post-command",
        action=example_post_command,
        run_for={"install", "create"},
    )
```

**conda_auth_handlers**() → collections.abc.Iterable[*conda.plugins.types.CondaAuthHandler*]

Register a conda auth handler derived from the requests API.

This plugin hook allows attaching requests auth handler subclasses, e.g. when authenticating requests against individual channels hosted at HTTP/HTTPS services.

**Example:**

```
import os
from conda import plugins
from requests.auth import AuthBase


class EnvironmentHeaderAuth(AuthBase):
    def __init__(self, *args, **kwargs):
```

(continues on next page)

(continued from previous page)

```
        self.username = os.environ["EXAMPLE_CONDA_AUTH_USERNAME"]
        self.password = os.environ["EXAMPLE_CONDA_AUTH_PASSWORD"]

    def __call__(self, request):
        request.headers["X-Username"] = self.username
        request.headers["X-Password"] = self.password
        return request


@plugins.hookimpl
def conda_auth_handlers():
    yield plugins.CondaAuthHandler(
        name="environment-header-auth",
        handler=EnvironmentHeaderAuth,
    )
```

**conda_health_checks**() → collections.abc.Iterable[*conda.plugins.types.CondaHealthCheck*]

Register health checks for conda doctor.

This plugin hook allows you to add more "health checks" to conda doctor that you can write to diagnose problems in your conda environment. Check out the health checks already shipped with conda for inspiration.

**Example:**

```
from conda import plugins


def example_health_check(prefix: str, verbose: bool):
    print("This is an example health check!")


@plugins.hookimpl
def conda_health_checks():
    yield plugins.CondaHealthCheck(
        name="example-health-check",
        action=example_health_check,
    )
```

**conda_pre_transaction_actions**() → collections.abc.Iterable[*conda.plugins.types.CondaPreTransactionAction*]

Register pre-transaction hooks.

Pre-transaction hooks run before all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

**Example:**

```
from conda import plugins
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
```

(continues on next page)

(continued from previous page)

```
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_pre_transaction_actions(
        self,
    ) -> Iterable[plugins.CondaPreTransactionAction]:
        yield plugins.CondaPreTransactionAction(
            name="example-pre-transaction-action",
            action=PrintAction,
        )
```

**conda_post_transaction_actions**() → collections.abc.Iterable[*conda.plugins.types.CondaPostTransactionAction*]

Register post-transaction hooks.

Post-transaction hooks run after all other actions run in a UnlinkLinkTransaction. For information about the Action class, see *Action*.

**Example:**

```
from conda import plugins
from conda.core.path_actions import Action


class PrintAction(Action):
    def verify(self):
        print("Performing verification...")
        self._verified = True

    def execute(self):
        print(
            self.transaction_context,
            self.target_prefix,
```

(continues on next page)

(continued from previous page)

```
            self.unlink_precs,
            self.link_precs,
            self.remove_specs,
            self.update_specs,
            self.neutered_specs,
        )

    def reverse(self):
        print("Reversing only happens when `execute` raises an exception.")

    def cleanup(self):
        print("Carrying out cleanup...")


class PrintActionPlugin:
    @plugins.hookimpl
    def conda_post_transaction_actions(
        self,
    ) -> Iterable[plugins.CondaPostTransactionAction]:
        yield plugins.CondaPostTransactionAction(
            name="example-post-transaction-action",
            action=PrintAction,
        )
```

**conda_pre_solves**() → collections.abc.Iterable[*conda.plugins.types.CondaPreSolve*]

Register pre-solve functions in conda that are used in the general solver API, before the solver processes the package specs in search of a solution.

**Example:**

```
from conda import plugins
from conda.models.match_spec import MatchSpec


def example_pre_solve(
    specs_to_add: frozenset[MatchSpec],
    specs_to_remove: frozenset[MatchSpec],
):
    print(f"Adding {len(specs_to_add)} packages")
    print(f"Removing {len(specs_to_remove)} packages")


@plugins.hookimpl
def conda_pre_solves():
    yield plugins.CondaPreSolve(
        name="example-pre-solve",
        action=example_pre_solve,
    )
```

**conda_post_solves**() → collections.abc.Iterable[*conda.plugins.types.CondaPostSolve*]

Register post-solve functions in conda that are used in the general solver API, after the solver has provided the package records to add or remove from the conda environment.

**Example:**

```python
from conda import plugins
from conda.models.records import PackageRecord


def example_post_solve(
    repodata_fn: str,
    unlink_precs: tuple[PackageRecord, ...],
    link_precs: tuple[PackageRecord, ...],
):
    print(f"Uninstalling {len(unlink_precs)} packages")
    print(f"Installing {len(link_precs)} packages")


@plugins.hookimpl
def conda_post_solves():
    yield plugins.CondaPostSolve(
        name="example-post-solve",
        action=example_post_solve,
    )
```

**conda_settings**() → collections.abc.Iterable[*conda.plugins.types.CondaSetting*]

Register new setting

The example below defines a simple string type parameter

**Example:**

```python
from conda import plugins
from conda.common.configuration import PrimitiveParameter, SequenceParameter


@plugins.hookimpl
def conda_settings():
    yield plugins.CondaSetting(
        name="example_option",
        description="This is an example option",
        parameter=PrimitiveParameter("default_value", element_type=str),
        aliases=("example_option_alias",),
    )
```

**conda_reporter_backends**() → collections.abc.Iterable[*conda.plugins.types.CondaReporterBackend*]

Register new reporter backend

The example below defines a reporter backend that uses the `pprint` module in Python.

**Example:**

```python
from pprint import pformat

from conda import plugins
from conda.plugins.types import (
    CondaReporterBackend,
    ReporterRendererBase,
    ProgressBarBase,
)
```

(continues on next page)

(continued from previous page)

```
class PprintReporterRenderer(ReporterRendererBase):
    "Implementation of the ReporterRendererBase"

    def detail_view(self, data):
        return pformat(data)

    def envs_list(self, data):
        formatted_data = pformat(data)
        return f"Environments: {formatted_data}"

    def progress_bar(self, description, io_context_manager) -> ProgressBarBase:
        "Returns our custom progress bar implementation"
        return PprintProgressBar(description, io_context_manager)


class PprintProgressBar(ProgressBarBase):
    "Blank implementation of ProgressBarBase which does nothing"

    def update_to(self, fraction) -> None:
        pass

    def refresh(self) -> None:
        pass

    def close(self) -> None:
        pass


@plugins.hookimpl
def conda_reporter_backends():
    yield CondaReporterBackend(
        name="pprint",
        description="Reporter backend based on the pprint module",
        renderer=PprintReporterRenderer,
    )
```

**conda_session_headers**(*host: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com`.

**Example:**

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}


@plugins.hookimpl
def conda_session_headers(host: str):
    if host in HOSTS:
```

(continues on next page)

(continued from previous page)

```
        yield plugins.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

**conda_request_headers**(*host: str*, *path: str*) → collections.abc.Iterable[*conda.plugins.types.CondaRequestHeader*]

Register new HTTP request headers

The example below defines how to add HTTP headers for all requests with the hostname of `example.com` and a `path/to/endpoint.json` path.

**Example:**

```
from conda import plugins

HOSTS = {"example.com", "sub.example.com"}
ENDPOINT = "/path/to/endpoint.json"


@plugins.hookimpl
def conda_request_headers(host: str, path: str):
    if host in HOSTS and path == ENDPOINT:
        yield plugins.CondaRequestHeader(
            name="Example-Header",
            value="example",
        )
```

**conda_prefix_data_loaders**() → collections.abc.Iterable[*conda.plugins.types.CondaPrefixDataLoader*]

Register new loaders for PrefixData

The example below defines how to expose the packages installed by a hypothetical 'penguin' tool as conda packages.

**Example:**

```
from pathlib import Path

from conda import plugins
from conda.common.path import PathType
from conda.models.records import PrefixRecord
from conda.plugins.types import CondaPrefixDataLoader


@plugins.hookimpl
def conda_prefix_data_loaders():
    yield CondaPrefixDataLoader(
        "hypothetical",
        load_hypothetical_packages,
    )


def load_hypothetical_packages(
    prefix: PathType, records: dict[str, PrefixRecord]
```

(continues on next page)

(continued from previous page)

```
) -> dict[str, PrefixRecord]:
    penguin_records = {}
    for info in Path(prefix).glob("lib/penguin/*.penguin-info"):
        name, version = info.name.rsplit("-", 1)
        kwargs = {}  # retrieve extra fields here
        penguin_records[name] = PrefixRecord(
            name=name, version=version, build_number=0, build="0", **kwargs
        )
    records.update(penguin_records)
    return penguin_records
```

**conda_environment_specifiers**() → collections.abc.Iterable[*conda.plugins.types.CondaEnvironmentSpecifier*]

**EXPERIMENTAL** Register new conda env spec type

The example below defines a type of conda env file called "random". It can parse a file with the file extension *.random*. This plugin will ignore whatever is in the input environment file and produce an environment with a random name and with random packages.

**Example:**

```
import json
import random
from pathlib import Path
from subprocess import run
from conda import plugins
from ...plugins.types import EnvironmentSpecBase
from conda.env.env import Environment

packages = ["python", "numpy", "scipy", "matplotlib", "pandas", "scikit-learn"]


class RandomSpec(EnvironmentSpecBase):
    extensions = {".random"}

    def __init__(self, filename: str):
        self.filename = filename

    def can_handle(self):
        # Return early if no filename was provided
        if self.filename is None:
            return False

        # Extract the file extension (e.g., '.txt' or '' if no extension)
        file_ext = os.path.splitext(self.filename)[1]

        # Check if the file has a supported extension and exists
        return any(
            spec_ext == file_ext and os.path.exists(self.filename)
            for spec_ext in RandomSpec.extensions
        )

    def env(self):
```

(continues on next page)

(continued from previous page)

```
        return Environment(
            name="".join(random.choice("0123456789abcdef") for i in range(6)),
            dependencies=[random.choice(packages) for i in range(6)],
        )


@plugins.hookimpl
def conda_environment_specifiers():
    yield plugins.CondaEnvSpec(
        name="random",
        environment_spec=RandomSpec,
    )
```

### manager

This module contains a subclass implementation of pluggy's PluginManager.

Additionally, it contains a function we use to construct the `PluginManager` object and register all plugins during conda's startup process.

## Classes

| | |
|---|---|
| *CondaPluginManager* | The conda plugin manager to implement behavior additional to pluggy's default plugin manager. |

## Functions

| | |
|---|---|
| *get_plugin_manager*(→ CondaPluginManager) | Get a cached version of the *CondaPluginManager* instance, |

**class CondaPluginManager**(*project_name: str | None = None*, **args*, ***kwargs*)

Bases: pluggy.PluginManager

The conda plugin manager to implement behavior additional to pluggy's default plugin manager.

**get_cached_solver_backend**

**get_canonical_name**(*plugin: object*) → str

Return a canonical name for a plugin object.

Note that a plugin may be registered under a different name specified by the caller of *register(plugin, name)*. To obtain the name of a registered plugin use `get_name(plugin)` instead.

**register**(*plugin*, *name: str | None = None*) → str | None

Call pluggy.PluginManager.register() and return the result or ignore errors raised, except `ValueError`, which means the plugin had already been registered.

**load_plugins**(**plugins*) → int

Load the provided list of plugins and fail gracefully on error. The provided list of plugins can either be classes or modules with *hookimpl*.

**load_entrypoints**(*group: str*, *name: str | None = None*) → int

Load modules from querying the specified setuptools `group`.

**Parameters**

- **group** (*str*) -- Entry point group to load plugins.
- **name** (*str*) -- If given, loads only plugins with the given `name`.

**Return type**
int

**Returns**
The number of plugins loaded by this call.

**get_hook_results**(*name: Literal[conda.plugins.subcommands]*) → list[*conda.plugins.types.CondaSubcommand*]

**get_hook_results**(*name: Literal[conda.plugins.virtual_packages]*) → list[*conda.plugins.types.CondaVirtualPackage*]

**get_hook_results**(*name: Literal[conda.plugins.solvers]*) → list[*conda.plugins.types.CondaSolver*]

**get_hook_results**(*name: Literal[pre_commands]*) → list[*conda.plugins.types.CondaPreCommand*]

**get_hook_results**(*name: Literal[post_commands]*) → list[*conda.plugins.types.CondaPostCommand*]

**get_hook_results**(*name: Literal[auth_handlers]*) → list[*conda.plugins.types.CondaAuthHandler*]

**get_hook_results**(*name: Literal[conda.plugins.subcommands.doctor.health_checks]*) → list[*conda.plugins.types.CondaHealthCheck*]

**get_hook_results**(*name: Literal[pre_solves]*) → list[*conda.plugins.types.CondaPreSolve*]

**get_hook_results**(*name: Literal[conda.plugins.post_solves]*) → list[*conda.plugins.types.CondaPostSolve*]

**get_hook_results**(*name: Literal[session_headers]*, ***, *host: str*) → list[*conda.plugins.types.CondaRequestHeader*]

**get_hook_results**(*name: Literal[request_headers]*, ***, *host: str*, *path: str*) → list[*conda.plugins.types.CondaRequestHeader*]

**get_hook_results**(*name: Literal[settings]*) → list[*conda.plugins.types.CondaSetting*]

**get_hook_results**(*name: Literal[conda.plugins.reporter_backends]*) → list[*conda.plugins.types.CondaReporterBackend*]

**get_hook_results**(*name: Literal[pre_transaction_actions]*) → list[*conda.plugins.types.CondaPreTransactionAction*]

**get_hook_results**(*name: Literal[post_transaction_actions]*) → list[*conda.plugins.types.CondaPostTransactionAction*]

**get_hook_results**(*name: Literal[conda.plugins.prefix_data_loaders]*) → list[*conda.plugins.types.CondaPrefixDataLoader*]

**get_hook_results**(*name: Literal[conda.plugins.environment_specifiers]*) → list[*conda.plugins.types.CondaEnvironmentSpecifier*]

Return results of the plugin hooks with the given name and raise an error if there is a conflict.

**get_solvers**() → dict[str, *conda.plugins.types.CondaSolver*]

Return a mapping from solver name to solver class.

**get_solver_backend**(*name: str | None = None*) → type[*conda.core.solve.Solver*]

Get the solver backend with the given name (or fall back to the name provided in the context).

See `context.solver` for more details.

Please use the cached version of this method called *`get_cached_solver_backend()`* for high-throughput code paths which is set up as a instance-specific LRU cache.

**get_auth_handler**(*name: str*) → type[requests.auth.AuthBase] | None

Get the auth handler with the given name or None

**get_settings**() → dict[str, *conda.plugins.types.CondaSetting*]

Return a mapping of plugin setting name to CondaSetting objects.

This method intentionally overwrites any duplicates that may be present

**invoke_pre_commands**(*command: str*) → None

Invokes `CondaPreCommand.action` functions registered with `conda_pre_commands`.

**Parameters**

**command** -- name of the command that is currently being invoked

**invoke_post_commands**(*command: str*) → None

Invokes `CondaPostCommand.action` functions registered with `conda_post_commands`.

**Parameters**

**command** -- name of the command that is currently being invoked

**disable_external_plugins**() → None

Disables all currently registered plugins except built-in conda plugins

**get_subcommands**() → dict[str, *conda.plugins.types.CondaSubcommand*]

**get_virtual_packages**() → tuple[*conda.plugins.types.CondaVirtualPackage*, Ellipsis]

**get_reporter_backends**() → tuple[*conda.plugins.types.CondaReporterBackend*, Ellipsis]

**get_reporter_backend**(*name: str*) → *conda.plugins.types.CondaReporterBackend*

Attempts to find a reporter backend while providing a fallback option if it is not found.

This method must return a valid `CondaReporterBackend` object or else it will raise an exception.

**get_virtual_package_records**() → tuple[*conda.models.records.PackageRecord*, Ellipsis]

**get_session_headers**(*host: str*) → dict[str, str]

**get_request_headers**(*host: str*, *path: str*) → dict[str, str]

**get_prefix_data_loaders**() → collections.abc.Iterable[conda.plugins.types.CondaPrefixDataLoaderCallable]

**invoke_health_checks**(*prefix: str*, *verbose: bool*) → None

**invoke_pre_solves**(*specs_to_add: frozenset[*conda.models.match_spec.MatchSpec*]*, *specs_to_remove: frozenset[*conda.models.match_spec.MatchSpec*]*) → None

Invokes `CondaPreSolve.action` functions registered with `conda_pre_solves`.

**Parameters**

- **specs_to_add**
- **specs_to_remove**

**invoke_post_solves**(*repodata_fn: str*, *unlink_precs: tuple[*conda.models.records.PackageRecord*, Ellipsis]*, *link_precs: tuple[*conda.models.records.PackageRecord*, Ellipsis]*) → None

Invokes `CondaPostSolve.action` functions registered with `conda_post_solves`.

**Parameters**

- **repodata_fn**
- **unlink_precs**
- **link_precs**

**load_settings**() → None

Iterates through all registered settings and adds them to the `conda.common.configuration.PluginConfig` class.

**get_config**(*data*) → *conda.plugins.config.PluginConfig*

Retrieve the configuration for the plugin. :returns: The configuration object for the plugin, initialized with raw data from the context. :rtype: PluginConfig

**get_environment_specifiers**() → dict[str, *conda.plugins.types.CondaEnvironmentSpecifier*]

Returns a mapping from environment specifier name to environment specifier.

**get_environment_specifier_by_name**(*source: str*, *name: str*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Get an environment specifier plugin by name

Raises PluginError if more than one environment_spec plugin is found to be able to handle the file. Raises CondaValueError if the requested plugin is not available.

**Parameters**

- **source** -- full path to the environment spec file/source
- **name** -- name of the environment plugin to load

**Returns**

an environment specifier plugin that matches the provided plugin name, or can handle the provided file

**detect_environment_specifier**(*source: str*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Detect the environment specifier plugin for a given spec source

Raises PluginError if more than one environment_spec plugin is found to be able to handle the file. Raises EnvironmentSpecPluginNotDetected if no plugins were found.

**Parameters**

**source** -- full path to the environment spec file or source

**Returns**

an environment specifier plugin that can handle the provided file

**get_environment_specifier**(*source: str*, *name: str = None*) → *conda.plugins.types.CondaEnvironmentSpecifier*

Get the environment specifier plugin for a given spec source, or given a plugin name Raises PluginError if more than one environment_spec plugin is found to be able to handle the file. Raises EnvironmentSpecPluginNotDetected if no plugins were found. Raises CondaValueError if the requested plugin is not available.

**Parameters**

- **filename** -- full path to the environment spec file/source
- **name** -- name of the environment plugin to load

**Returns**
an environment specifier plugin that matches the provided plugin name, or can handle the provided file

`get_pre_transaction_actions`(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*) → list[*conda.core.path_actions.Action*]

Get the plugin-defined pre-transaction actions.

**Parameters**

- **`transaction_context`** -- Mapping between target prefixes and PrefixActions instances
- **`target_prefix`** -- Target prefix for the action
- **`unlink_precs`** -- Package records to be unlinked
- **`link_precs`** -- Package records to link
- **`remove_specs`** -- Specs to be removed
- **`update_specs`** -- Specs to be updated
- **`neutered_specs`** -- Specs to be neutered

**Returns**
The plugin-defined pre-transaction actions

`get_post_transaction_actions`(*transaction_context: dict[str, str] | None = None*, *target_prefix: str | None = None*, *unlink_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *link_precs: collections.abc.Iterable[conda.models.records.PackageRecord] | None = None*, *remove_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *update_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*, *neutered_specs: collections.abc.Iterable[conda.models.match_spec.MatchSpec] | None = None*) → list[*conda.core.path_actions.Action*]

Get the plugin-defined post-transaction actions.

**Parameters**

- **`transaction_context`** -- Mapping between target prefixes and PrefixActions instances
- **`target_prefix`** -- Target prefix for the action
- **`unlink_precs`** -- Package records to be unlinked
- **`link_precs`** -- Package records to link
- **`remove_specs`** -- Specs to be removed

- **update_specs** -- Specs to be updated
- **neutered_specs** -- Specs to be neutered

**Returns**
The plugin-defined post-transaction actions

**get_plugin_manager**() → *CondaPluginManager*

Get a cached version of the *CondaPluginManager* instance, with the built-in and entrypoints provided by the plugins loaded.

### post_solves

Register the built-in post_solves hook implementations.

### signature_verification

Register signature verification as a post-solve plugin.

## Functions

*conda_post_solves*()

**conda_post_solves**()

**plugins**

### prefix_data_loaders

### pypi

Reads PyPI packages in a conda prefix that have been installed with non-conda tools.

### pkg_format

Common Python package format utilities.

### Classes

| | |
|---|---|
| *PythonDistribution* | Base object describing a python distribution based on path to anchor file. |
| *PythonInstalledDistribution* | Python distribution installed via distutils. |
| *PythonEggInfoDistribution* | Python distribution installed via setuptools. |
| *PythonEggLinkDistribution* | Python distribution installed via setuptools. |
| *PythonDistributionMetadata* | Object representing the metada of a Python Distribution given by anchor |
| *Evaluator* | This class is used to evaluate marker expressions. |

### Functions

| | |
|---|---|
| *norm_package_name*(name) | |
| *pypi_name_to_conda_name*(pypi_name) | |
| *norm_package_version*(version) | Normalize a version by removing extra spaces and parentheses. |
| *split_spec*(spec, sep) | Split a spec by separator and return stripped start and end parts. |
| *parse_specification*(spec) | Parse a requirement from a python distribution metadata and return a |
| *get_site_packages_anchor_files*(site_packages_pa ...) | Get all the anchor files for the site packages directory. |
| *get_dist_file_from_egg_link*(egg_link_file, prefix_path) | Return the egg info file path following an egg link. |
| *parse_marker*(marker_string) | Parse marker string and return a dictionary containing a marker expression. |
| *_is_literal*(o) | |
| *get_default_marker_context*() | Return the default context dictionary to use when parsing markers. |
| *interpret*(marker[, execution_context]) | Interpret a marker and return a result depending on environment. |
| *read_python_record*(prefix_path, anchor_file, ...) | Convert a python package defined by an anchor file (Metadata information) |

### Attributes

| |
|---|
| *PYPI_TO_CONDA* |
| *PYPI_CONDA_DEPS* |
| *PARTIAL_PYPI_SPEC_PATTERN* |
| *PY_FILE_RE* |
| *PySpec* |
| *IDENTIFIER* |
| *VERSION_IDENTIFIER* |
| *COMPARE_OP* |
| *MARKER_OP* |
| *OR* |
| *AND* |
| *NON_SPACE* |
| *STRING_CHUNK* |
| *DEFAULT_MARKER_CONTEXT* |
| *evaluator* |

**PYPI_TO_CONDA**

**PYPI_CONDA_DEPS**

**PARTIAL_PYPI_SPEC_PATTERN**

**PY_FILE_RE**

**PySpec**

**exception MetadataWarning**

Bases: Warning

Base class for warning categories.

Initialize self. See help(type(self)) for accurate signature.

**class PythonDistribution**(*anchor_full_path*, *python_version*)

Base object describing a python distribution based on path to anchor file.

**property name**

property norm_name

property conda_name

property version

MANIFEST_FILES = ()

REQUIRES_FILES = ()

MANDATORY_FILES = ()

ENTRY_POINTS_FILES = ('entry_points.txt',)

static init(*prefix_path*, *anchor_file*, *python_version*)

_check_files()
Check the existence of mandatory files for a given distribution.

_check_path_data(*path*, *checksum*, *size*)
Normalizes record data content and format.

static _parse_requires_file_data(*data*, *global_section='__global__'*)

static _parse_entries_file_data(*data*)

_load_requires_provides_file()

manifest_full_path()

get_paths()
Read the list of installed paths from record or source file.

#### Example

**[(u'skdata/__init__.py', u'sha256=47DEQpj8HBSa-_TImW-5JCeuQeRkm5NMpJWZG3hSuFU', 0),**
(u'skdata/diabetes.py', None, None), ...

]

get_dist_requirements()

get_python_requirements()

get_external_requirements()

get_extra_provides()

get_conda_dependencies()
Process metadata fields providing dependency information.

This includes normalizing fields, and evaluating environment markers.

abstract get_optional_dependencies()

get_entry_points()

class PythonInstalledDistribution(*prefix_path*, *anchor_file*, *python_version*)
Bases: *PythonDistribution*

Python distribution installed via distutils.

### Notes

- https://www.python.org/dev/peps/pep-0376/

MANIFEST_FILES = ('RECORD',)

REQUIRES_FILES = ()

MANDATORY_FILES = ('METADATA',)

ENTRY_POINTS_FILES = ()

is_manageable = True

class PythonEggInfoDistribution(*anchor_full_path*, *python_version*, *sp_reference*)

Bases: *PythonDistribution*

Python distribution installed via setuptools.

### Notes

- http://peak.telecommunity.com/DevCenter/EggFormats

property is_manageable

MANIFEST_FILES = ('installed-files.txt', 'SOURCES', 'SOURCES.txt')

REQUIRES_FILES = ('requires.txt', 'depends.txt')

MANDATORY_FILES = ()

ENTRY_POINTS_FILES = ('entry_points.txt',)

class PythonEggLinkDistribution(*prefix_path*, *anchor_file*, *python_version*)

Bases: *PythonEggInfoDistribution*

Python distribution installed via setuptools.

### Notes

- http://peak.telecommunity.com/DevCenter/EggFormats

is_manageable = False

class PythonDistributionMetadata(*path*)

Object representing the metada of a Python Distribution given by anchor file (or directory) path.

This metadata is extracted from a single file. Python distributions might create additional files that complement this metadata information, but that is handled at the python distribution level.

### Notes

- https://packaging.python.org/specifications/core-metadata/
- Metadata 2.1: https://www.python.org/dev/peps/pep-0566/
- Metadata 2.0: https://www.python.org/dev/peps/pep-0426/ (Withdrawn)
- Metadata 1.2: https://www.python.org/dev/peps/pep-0345/
- Metadata 1.1: https://www.python.org/dev/peps/pep-0314/
- Metadata 1.0: https://www.python.org/dev/peps/pep-0241/

**property name**

**property version**

**FILE_NAMES = ('METADATA', 'PKG-INFO')**

**SINGLE_USE_KEYS**

**MULTIPLE_USE_KEYS**

**static _process_path**(*path*, *metadata_filenames*)

Find metadata file inside dist-info folder, or check direct file.

**classmethod _message_to_dict**(*message*)

Convert the RFC-822 headers data into a dictionary.

*message* is an email.parser.Message instance.

The canonical method to transform metadata fields into such a data structure is as follows:

- The original key-value format should be read with email.parser.HeaderParser
- All transformed keys should be reduced to lower case. Hyphens should be replaced with underscores, but otherwise should retain all other characters
- The transformed value for any field marked with "(Multiple-use") should be a single list containing all the original values for the given key
- The Keywords field should be converted to a list by splitting the original value on whitespace characters
- The message body, if present, should be set to the value of the description key.
- The result should be stored as a string-keyed dictionary.

**classmethod _read_metadata**(*fpath*)

Read the original format which is stored as RFC-822 headers.

**_get_multiple_data**(*keys*)

Helper method to get multiple data values by keys.

Keys is an iterable including the preferred key in order, to include values of key that might have been replaced (deprecated), for example keys can be ['requires_dist', 'requires'], where the key 'requires' is deprecated and replaced by 'requires_dist'.

**get_dist_requirements**()

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming some other distutils project required by this distribution.

**The format of a requirement string contains from one to four parts:**

- A project name, in the same format as the Name: field. The only mandatory part.
- A comma-separated list of 'extra' names. These are defined by the required project, referring to specific features which may need extra dependencies.
- A version specifier. Tools parsing the format should accept optional parentheses around this, but tools generating it should not use parentheses.
- An environment marker after a semicolon. This means that the requirement is only needed in the specified conditions.

This field may be followed by an environment marker after a semicolon.

### Example

**frozenset(['pkginfo', 'PasteDeploy', 'zope.interface (>3.5.0)',**
'pywin32 >1.0; sys_platform == "win32"'])

Return 'Requires' if 'Requires-Dist' is empty.

`get_python_requirements()`

New in version 1.2.

This field specifies the Python version(s) that the distribution is guaranteed to be compatible with. Installation tools may look at this when picking which version of a project to install.

The value must be in the format specified in Version specifiers.

This field may be followed by an environment marker after a semicolon.

### Example

**frozenset(['>=3', '>2.6,!=3.0.*,!=3.1.*', '~=2.6',**
'>=3; sys_platform == "win32"'])

`get_external_requirements()`

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing some dependency in the system that the distribution is to be used. This field is intended to serve as a hint to downstream project maintainers, and has no semantics which are meaningful to the distutils distribution.

The format of a requirement string is a name of an external dependency, optionally followed by a version declaration within parentheses.

This field may be followed by an environment marker after a semicolon.

Because they refer to non-Python software releases, version numbers for this field are not required to conform to the format specified in PEP 440: they should correspond to the version scheme used by the external dependency.

Notice that there's is no particular rule on the strings to be used!

### Example

frozenset(['C', 'libpng (>=1.5)', 'make; sys_platform != "win32"'])

`get_extra_provides()`

New in version 2.1.

A string containing the name of an optional feature. Must be a valid Python identifier. May be used to make a dependency conditional on hether the optional feature has been requested.

### Example

frozenset(['pdf', 'doc', 'test'])

`get_dist_provides()`

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming a Distutils project which is contained within this distribution. This field must include the project identified in the Name field, followed by the version : Name (Version).

A distribution may provide additional names, e.g. to indicate that multiple projects have been bundled together. For instance, source distributions of the ZODB project have historically included the transaction project, which is now available as a separate distribution. Installing such a source distribution satisfies requirements for both ZODB and transaction.

A distribution may also provide a "virtual" project name, which does not correspond to any separately-distributed project: such a name might be used to indicate an abstract capability which could be supplied by one of multiple projects. E.g., multiple projects might supply RDBMS bindings for use by a given ORM: each project might declare that it provides ORM-bindings, allowing other projects to depend only on having at most one of them installed.

A version declaration may be supplied and must follow the rules described in Version specifiers. The distribution's version number will be implied if none is specified.

This field may be followed by an environment marker after a semicolon.

Return *Provides* in case *Provides-Dist* is empty.

`get_dist_obsolete()`

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing a distutils project's distribution which this distribution renders obsolete, meaning that the two projects should not be installed at the same time.

Version declarations can be supplied. Version numbers must be in the format specified in Version specifiers [1].

The most common use of this field will be in case a project name changes, e.g. Gorgon 2.3 gets subsumed into Torqued Python 1.0. When you install Torqued Python, the Gorgon distribution should be removed.

This field may be followed by an environment marker after a semicolon.

Return *Obsoletes* in case *Obsoletes-Dist* is empty.

#### Example

frozenset(['Gorgon', "OtherProject (<3.0) ; python_version == '2.7'"])

#### Notes

- [1] https://packaging.python.org/specifications/version-specifiers/

`get_classifiers`()

Classifiers are described in PEP 301, and the Python Package Index publishes a dynamic list of currently defined classifiers.

This field may be followed by an environment marker after a semicolon.

#### Example

**frozenset(['Development Status :: 4 - Beta',**
"Environment :: Console (Text Based) ; os_name == "posix"])

`norm_package_name`(*name*)

`pypi_name_to_conda_name`(*pypi_name*)

`norm_package_version`(*version*)

Normalize a version by removing extra spaces and parentheses.

`split_spec`(*spec*, *sep*)

Split a spec by separator and return stripped start and end parts.

`parse_specification`(*spec*)

Parse a requirement from a python distribution metadata and return a namedtuple with name, extras, constraints, marker and url components.

This method does not enforce strict specifications but extracts the information which is assumed to be *correct*. As such no errors are raised.

#### Example

**PySpec(name='requests', extras=['security'], constraints='>=3.3.0',**
marker='foo >= 2.7 or bar == 1', url='])

`get_site_packages_anchor_files`(*site_packages_path*, *site_packages_dir*)

Get all the anchor files for the site packages directory.

`get_dist_file_from_egg_link`(*egg_link_file*, *prefix_path*)

Return the egg info file path following an egg link.

`parse_marker`(*marker_string*)

Parse marker string and return a dictionary containing a marker expression.

The dictionary will contain keys "op", "lhs" and "rhs" for non-terminals in the expression grammar, or strings. A string contained in quotes is to be interpreted as a literal string, and a string not contained in quotes is a variable (such as os_name).

**IDENTIFIER**

**VERSION_IDENTIFIER**

**COMPARE_OP**

**MARKER_OP**

**OR**

**AND**

**NON_SPACE**

**STRING_CHUNK**

**_is_literal**(*o*)

**class Evaluator**

This class is used to evaluate marker expressions.

**operations**

**evaluate**(*expr*, *context*)

Evaluate a marker expression returned by the `parse_requirement()` function in the specified context.

**get_default_marker_context**()

Return the default context dictionary to use when parsing markers.

**DEFAULT_MARKER_CONTEXT**

**evaluator**

**interpret**(*marker*, *execution_context=None*)

Interpret a marker and return a result depending on environment.

**Parameters**

- **marker** (*str*) -- The marker to interpret.
- **execution_context** (*mapping*) -- The context used for name lookup.

**read_python_record**(*prefix_path*, *anchor_file*, *python_version*)

Convert a python package defined by an anchor file (Metadata information) into a conda prefix record object.

## Classes

| | |
|---|---|
| *PrefixGraph* | A directed graph structure used for sorting packages (prefix_records) in prefixes and |
| *CondaPrefixDataLoader* | Define new loaders to expose non-conda packages in a given prefix |

### Functions

| | |
|---|---|
| get_python_site_packages_short_path(python_ve | |
| win_path_ok(path) | |
| rm_rf(→ bool) | Completely delete path |
| get_site_packages_anchor_files(site_packages_pa ...) | Get all the anchor files for the site packages directory. |
| read_python_record(prefix_path, anchor_file, ...) | Convert a python package defined by an anchor file (Metadata information) |
| load_site_packages(→ dict[str, ...) | Load non-conda-installed python packages in the site-packages of the prefix. |
| get_conda_anchor_files_and_records(...) | Return the anchor files for the conda records of python packages. |
| conda_prefix_data_loaders() | |

### Attributes

| | |
|---|---|
| hookimpl | Decorator used to mark plugin hook implementations |

**exception ValidationError**(*key*, *value=None*, *valid_types=None*, *msg=None*)

Bases: AuxlibError, TypeError

Mixin to identify exceptions associated with the auxlib package.

Initialize self. See help(type(self)) for accurate signature.

**get_python_site_packages_short_path**(*python_version*)

**win_path_ok**(*path*)

**rm_rf**(*path: str | os.PathLike*, *clean_empty_parents: bool = False*) → bool

Completely delete path max_retries is the number of times to retry on failure. The default is 5. This only applies to deleting a directory. If removing path fails and trash is True, files will be moved to the trash directory.

**class PrefixGraph**(*records*, *specs=()*)

A directed graph structure used for sorting packages (prefix_records) in prefixes and manipulating packages within prefixes (e.g. removing and pruning).

The terminology used for edge direction is "parents" and "children" rather than "successors" and "predecessors". The parent nodes of a record are those records in the graph that match the record's "depends" field. E.g. NodeA depends on NodeB, then NodeA is a child of NodeB, and NodeB is a parent of NodeA. Nodes can have zero parents, or more than two parents.

Most public methods mutate the graph.

**property records**

**remove_spec**(*spec*)

Remove all matching nodes, and any associated child nodes.

**Parameters**
**spec** (MatchSpec)

**Returns**
The removed nodes.

**Return type**
tuple[*PrefixRecord*]

**remove_youngest_descendant_nodes_with_specs()**

A specialized method used to determine only dependencies of requested specs.

**Returns**
The removed nodes.

**Return type**
tuple[*PrefixRecord*]

**prune()**

Prune back all packages until all child nodes are anchored by a spec.

**Returns**
The pruned nodes.

**Return type**
tuple[*PrefixRecord*]

**get_node_by_name**(*name*)

**all_descendants**(*node*)

**all_ancestors**(*node*)

**_remove_node**(*node*)

Removes this node and all edges referencing it.

**_toposort()**

**classmethod _toposort_raise_on_cycles**(*graph*)

**classmethod _topo_sort_handle_cycles**(*graph*)

**static _toposort_pop_key**(*graph*)

Pop an item from the graph that has the fewest parents. In the case of a tie, use the node with the alphabetically-first package name.

**static _toposort_prepare_graph**(*graph*)

**hookimpl**

Decorator used to mark plugin hook implementations

**class CondaPrefixDataLoader**

Define new loaders to expose non-conda packages in a given prefix as PrefixRecord objects.

**Parameters**

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to PrefixRecord objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

**name:** str

**loader:** CondaPrefixDataLoaderCallable

**get_site_packages_anchor_files**(*site_packages_path*, *site_packages_dir*)

Get all the anchor files for the site packages directory.

**read_python_record**(*prefix_path*, *anchor_file*, *python_version*)

Convert a python package defined by an anchor file (Metadata information) into a conda prefix record object.

**load_site_packages**(*prefix: conda.common.path.PathType*, *records: dict[str, conda.models.records.PrefixRecord]*) → dict[str, *conda.models.records.PrefixRecord*]

Load non-conda-installed python packages in the site-packages of the prefix.

Python packages not handled by conda are installed via other means, like using pip or using python setup.py develop for local development.

Packages found that are not handled by conda are converted into a prefix record and handled in memory.

Packages clobbering conda packages (i.e. the conda-meta record) are removed from the in memory representation.

**get_conda_anchor_files_and_records**(*site_packages_short_path*, *python_records*)

Return the anchor files for the conda records of python packages.

**conda_prefix_data_loaders**()

**plugins**

### reporter_backends

### console

Defines a "console" reporter backend.

This reporter backend provides the default output for conda.

## Classes

| | |
|---|---|
| *QuietProgressBar* | Progress bar class used when no output should be printed |
| *TQDMProgressBar* | Progress bar class used for tqdm progress bars |
| *Spinner* | Helper class that provides a standard way to create an ABC using |
| *QuietSpinner* | Helper class that provides a standard way to create an ABC using |
| *ConsoleReporterRenderer* | Default implementation for console reporting in conda |

### Functions

| | |
|---|---|
| conda_reporter_backends() | Reporter backend for console |

**class QuietProgressBar**(*description: str*, ***kwargs*)

Bases: conda.plugins.types.ProgressBarBase

Progress bar class used when no output should be printed

**update_to**(*fraction*) → None

**refresh**() → None

**close**() → None

**class TQDMProgressBar**(*description: str*, *position=None*, *leave=True*, ***kwargs*)

Bases: conda.plugins.types.ProgressBarBase

Progress bar class used for tqdm progress bars

**update_to**(*fraction*) → None

**close**() → None

**refresh**() → None

**static _tqdm**(**args*, ***kwargs*)

Deferred import so it doesn't hit the *conda activate* paths.

**class Spinner**(*message*, *fail_message='failed\n'*)

Bases: conda.plugins.types.SpinnerBase

Helper class that provides a standard way to create an ABC using inheritance.

**spinner_cycle**

**start**()

**stop**()

**_start_spinning**()

**__enter__**()

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)

**class QuietSpinner**(*message: str*, *fail_message: str = 'failed\n'*)

Bases: conda.plugins.types.SpinnerBase

Helper class that provides a standard way to create an ABC using inheritance.

**__enter__**()

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)

**class ConsoleReporterRenderer**

Bases: conda.plugins.types.ReporterRendererBase

Default implementation for console reporting in conda

**detail_view**(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

**static envs_list**(*prefixes: collections.abc.Iterable[conda.common.path.PathType]*, *output=True*, ***kwargs*) → str

Render a list of environments

**progress_bar**(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Determines whether to return a TQDMProgressBar or QuietProgressBar

**spinner**(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Determines whether to return a Spinner or QuietSpinner

**prompt**(*message='Proceed'*, *choices=('yes', 'no')*, *default='yes'*) → str

Implementation of a prompt dialog

**conda_reporter_backends**()

Reporter backend for console

### json

Defines a JSON reporter backend

This reporter backend is used to provide JSON strings for output rendering. It is essentially just a wrapper around `conda.common.serialize.json_dump`.

## Classes

| | |
|---|---|
| *JSONProgressBar* | Progress bar that outputs JSON to stdout |
| *JSONReporterRenderer* | Default implementation for JSON reporting in conda |
| *JSONSpinner* | This class for a JSONSpinner does nothing because we do not want to include this output. |

## Functions

| | |
|---|---|
| *conda_reporter_backends*() | Reporter backend for JSON |

**class JSONProgressBar**(*description: str*, *enabled: bool = True*, ***kwargs*)

Bases: *conda.plugins.types.ProgressBarBase*

Progress bar that outputs JSON to stdout

**update_to**(*fraction*) → None

**refresh**()

**close**()

**classmethod get_lock**()

Used for our own sys.stdout.write/flush calls

**class JSONReporterRenderer**

Bases: *conda.plugins.types.ReporterRendererBase*

Default implementation for JSON reporting in conda

**render**(*data: Any*, ***kwargs*) → str

**detail_view**(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

**envs_list**(*data*, ***kwargs*) → str

Render a list of environments

**progress_bar**(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Return a ProgressBarBase~ object to use as a progress bar

**spinner**(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Return a SpinnerBase~ object to use as a spinner (i.e. loading dialog)

**prompt**(*message: str = 'Proceed'*, *choices=('yes', 'no')*, *default: str = 'yes'*) → str

For this class, we want this method to do nothing

**class JSONSpinner**(*message: str*, *fail_message: str = 'failed\n'*)

Bases: *conda.plugins.types.SpinnerBase*

This class for a JSONSpinner does nothing because we do not want to include this output.

**__enter__**()

**__exit__**(*exc_type*, *exc_val*, *exc_tb*)

**conda_reporter_backends**()

Reporter backend for JSON

This is the default reporter backend that returns objects as JSON strings.

**plugins**

### solvers

Register the classic conda solver.

## Functions

| | |
|---|---|
| *conda_solvers*() | The classic solver as shipped by default in conda. |

**conda_solvers**()

The classic solver as shipped by default in conda.

### subcommands

### doctor

Implementation for *conda doctor* subcommand. Adds various environment and package checks to detect issues or possible environment corruption.

### health_checks

Backend logic implementation for *conda doctor*.

## Functions

| | |
|---|---|
| *check_envs_txt_file*(→ bool) | Checks whether the environment is listed in the environments.txt file |
| *excluded_files_check*(→ bool) | |
| *find_packages_with_missing_files*(→ dict[str, list[str]]) | Finds packages listed in conda-meta which have missing files. |
| *find_altered_packages*(→ dict[str, list[str]]) | Finds altered packages |
| *missing_files*(→ None) | |
| *altered_files*(→ None) | |
| *env_txt_check*(→ None) | |
| *requests_ca_bundle_check*(→ None) | |
| *consistent_env_check*(→ None) | |
| *conda_health_checks*() | |

## Attributes

| |
|---|
| *logger* |
| *OK_MARK* |
| *X_MARK* |

**logger**

**OK_MARK = ''**

**X_MARK = ''**

**check_envs_txt_file**(*prefix: str | os.PathLike | pathlib.Path*) → bool

Checks whether the environment is listed in the environments.txt file

**excluded_files_check**(*filename: str*) → bool

**find_packages_with_missing_files**(*prefix: str | pathlib.Path*) → dict[str, list[str]]

Finds packages listed in conda-meta which have missing files.

**find_altered_packages**(*prefix: str | pathlib.Path*) → dict[str, list[str]]

Finds altered packages

**missing_files**(*prefix: str*, *verbose: bool*) → None

**altered_files**(*prefix: str*, *verbose: bool*) → None

**env_txt_check**(*prefix: str*, *verbose: bool*) → None

**requests_ca_bundle_check**(*prefix: str*, *verbose: bool*) → None

**consistent_env_check**(*prefix: str*, *verbose: bool*) → None

**conda_health_checks**()

### Classes

| | |
|---|---|
| *PrefixData* | The PrefixData class aims to be the representation of the state |
| *CondaSubcommand* | Return type to use when defining a conda subcommand plugin hook. |

### Functions

| | |
|---|---|
| *add_parser_help*(→ None) | So we can use consistent capitalization and periods in the help. You must |
| *add_parser_prefix*(→ parse._MutuallyExclusiveGroup) | arg- |
| *add_parser_verbose*(→ None) | |
| *configure_parser*(parser) | |
| *execute*(→ None) | Run registered health_check plugins. |
| *conda_subcommands*() | |

### Attributes

| | |
|---|---|
| *context* | |
| *hookimpl* | Decorator used to mark plugin hook implementations |

**context**

**add_parser_help**(*p: argparse.ArgumentParser*) → None

So we can use consistent capitalization and periods in the help. You must use the add_help=False argument to ArgumentParser or add_parser to use this. Add this first to be consistent with the default argparse output.

**add_parser_prefix**(*p: argparse.ArgumentParser*, *prefix_required: bool = False*) → argparse._MutuallyExclusiveGroup

**add_parser_verbose**(*parser: argparse.ArgumentParser | argparse._ArgumentGroup*) → None

**class PrefixData**(*prefix_path: str | os.PathLike[str] | pathlib.Path*, *interoperability: bool | None = None*)

The PrefixData class aims to be the representation of the state of a conda environment on disk. The directory where the environment lives is called prefix.

This class supports different types of tasks:

- Reading and querying *conda-meta/*.json* files as *PrefixRecord* objects
- Reading and writing environment-specific configuration (env vars, state file, nonadmin markers, etc)
- Existence checks and validations of name, path, and magic files / markers
- Exposing non-conda packages installed in prefix as *PrefixRecord*, via the plugin system

**property name: str**

Returns the name of the environment, if available.

If the environment doesn't live in one the configured *envs_dirs*, an empty string is returned. The construct *prefix_data.name or prefix_data.prefix_path* can be helpful in those cases.

**property is_writable: bool | None | conda.auxlib._Null**

Check whether the configured path is writable. This is assessed by checking whether *conda-meta/history* is writable. It if is, it is assumed that the rest of the directory tree is writable too.

Note: The value is cached in the instance. Use *.assert_writable()* for a non- cached check.

**property _pip_interop_enabled**

**property _prefix_records: dict[str, *conda.models.records.PrefixRecord*] | None**

**property _python_pkg_record: *conda.models.records.PrefixRecord* | None**

Return the prefix record for the package python.

**_cache_: dict[tuple[pathlib.Path, bool | None], *PrefixData*]**

**classmethod from_name**(*name: str*, ***kwargs*) → *PrefixData*

Creates a PrefixData instance from an environment name.

The name will be validated with *PrefixData.validate_name()* if it does not exist.

**Parameters**

**name** -- The name of the environment. Must not contain path separators (/, ).

**Raises**
*CondaValueError* -- If *name* contains a path separator.

**classmethod from_context**(*validate: bool = False*) → *PrefixData*

Creates a PrefixData instance from the path specified by *context.target_prefix*.

The path and name will be validated with *PrefixData.validate_path()* and *PrefixData.validate_name()*, respectively, if *validate* is *True*.

**Parameters**
**validate** -- Whether the path and name should be validated. Useful for environments about to be created.

**__eq__**(*other: Any*) → bool

Return self==value.

**exists**() → bool

Check whether the PrefixData path exists and is a directory.

**is_environment**() → bool

Check whether the PrefixData path is a valida conda environment.

This is assessed by checking if *conda-meta/history* marker file exists.

**is_frozen**() → bool

Check whether the environment is marked as frozen, as per CEP 22.

This is assessed by checking if *conda-meta/frozen* marker file exists.

**is_base**() → bool

Check whether the configured path refers to the *base* environment.

**assert_exists**() → None

Check whether the environment path exists.

**Raises**
*EnvironmentLocationNotFound* -- If the check returns False.

**assert_environment**() → None

Check whether the environment path exists and is a valid conda environment.

**Raises**
*DirectoryNotACondaEnvironmentError* -- If the check returns False.

**assert_writable**() → None

Check whether the environment path is a valid conda environment and is writable.

**Raises**
*EnvironmentNotWritableError* -- If the check returns False.

**assert_not_frozen**() → None

Check whether the environment path is a valid conda environment and is not marked as frozen (as per CEP 22).

**Raises**
*EnvironmentIsFrozenError* -- If the environment is marked as frozen.

**validate_path**(*expand_path: bool = False*) → None

Validate the path of the environment.

It runs the following checks:

- Make sure the path does not contain *:* or *;* (OS-dependent).
- Disallow immediately nested environments (e.g. *$CONDA_ROOT* and *$CONDA_ROOT/my-env*).
- Warn if there are spaces in the path.

**Parameters**
**expand_path** -- Whether to process ~ and environment variables in the string. The expanded value will replace *.prefix_path*.

**Raises**
***CondaValueError*** -- If the environment contains *:*, *;*, or is nested.

**validate_name**(*allow_base: bool = False*) → None

Validate the name of the environment.

**Parameters**
**allow_base** -- Whether to allow *base* as a valid name.

**Raises**
***CondaValueError*** -- If the name is protected, or if it contains disallowed characters (*/*, *`* , *`:*, *#*).

**load**() → None

**reload**() → *PrefixData*

**_get_json_fn**(*prefix_record:* conda.models.records.PrefixRecord) → str

**insert**(*prefix_record:* conda.models.records.PrefixRecord, *remove_auth: bool = True*) → None

**remove**(*package_name: str*) → None

**get**(*package_name: str*, *default: T = NULL*) → *conda.models.records.PackageRecord* | T

**iter_records**() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**iter_records_sorted**() → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**all_subdir_urls**() → set[str]

**query**(*package_ref_or_match_spec:* conda.models.records.PackageRecord | conda.models.match_spec.MatchSpec | *str*) → collections.abc.Iterable[*conda.models.records.PrefixRecord*]

**_load_single_record**(*prefix_record_json_path: conda.common.path.PathType*) → None

**_load_site_packages**() → dict[str, *conda.models.records.PrefixRecord*]

**_get_environment_state_file**() → dict[str, dict[str, str]]

**_write_environment_state_file**(*state: dict[str, dict[str, str]]*) → None

**get_environment_env_vars**() → dict[str, str] | dict[bytes, bytes]

**set_environment_env_vars**(*env_vars: dict[str, str]*) → dict[str, str] | None

**unset_environment_env_vars**(*env_vars: dict[str, str]*) → dict[str, str] | None

**set_nonadmin**() → None

Creates $PREFIX/.nonadmin if sys.prefix/.nonadmin exists (on Windows).

**class CondaSubcommand**

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

**Parameters**

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

**name:** str

**summary:** str

**action:** Callable[[argparse.Namespace | tuple[str]], int | None]

**configure_parser:** Callable[[argparse.ArgumentParser], None] | None

**hookimpl**

Decorator used to mark plugin hook implementations

**configure_parser**(*parser: argparse.ArgumentParser*)

**execute**(*args: argparse.Namespace*) → None

Run registered health_check plugins.

**conda_subcommands**()

**plugins**

### types

Definition of specific return types for use when defining a conda plugin hook.

Each type corresponds to the plugin hook for which it is used.

### Classes

| | |
|---|---|
| *CondaSubcommand* | Return type to use when defining a conda subcommand plugin hook. |
| *CondaVirtualPackage* | Return type to use when defining a conda virtual package plugin hook. |
| *CondaSolver* | Return type to use when defining a conda solver plugin hook. |
| *CondaPreCommand* | Return type to use when defining a conda pre-command plugin hook. |
| *CondaPostCommand* | Return type to use when defining a conda post-command plugin hook. |
| *ChannelNameMixin* | Class mixin to make all plugin implementations compatible, e.g. when they |
| *ChannelAuthBase* | Base class that we require all plugin implementations to use to be compatible. |
| *CondaAuthHandler* | Return type to use when the defining the conda auth handlers hook. |
| *CondaHealthCheck* | Return type to use when defining conda health checks plugin hook. |
| *CondaPreSolve* | Return type to use when defining a conda pre-solve plugin hook. |
| *CondaPostSolve* | Return type to use when defining a conda post-solve plugin hook. |
| *CondaSetting* | Return type to use when defining a conda setting plugin hook. |
| *ProgressBarBase* | Helper class that provides a standard way to create an ABC using |
| *SpinnerBase* | Helper class that provides a standard way to create an ABC using |
| *ReporterRendererBase* | Base class for all reporter renderers. |
| *CondaReporterBackend* | Return type to use when defining a conda reporter backend plugin hook. |
| *CondaRequestHeader* | Define vendor specific headers to include HTTP requests |
| *CondaPreTransactionAction* | Return type to use when defining a pre-transaction action hook. |
| *CondaPostTransactionAction* | Return type to use when defining a post-transaction action hook. |
| *CondaPrefixDataLoader* | Define new loaders to expose non-conda packages in a given prefix |
| *EnvironmentSpecBase* | **EXPERIMENTAL** |
| *CondaEnvironmentSpecifier* | **EXPERIMENTAL** |

### Attributes

| |
|---|
| *CondaPrefixDataLoaderCallable* |

**CondaPrefixDataLoaderCallable: TypeAlias**

**class CondaSubcommand**

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

**Parameters**

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

**name: str**

**summary: str**

**action: Callable[[argparse.Namespace | tuple[str]], int | None]**

**configure_parser: Callable[[argparse.ArgumentParser], None] | None**

**class CondaVirtualPackage**

Bases: `NamedTuple`

Return type to use when defining a conda virtual package plugin hook.

For details on how this is used, see *conda_virtual_packages()*.

**Parameters**

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).
- **build** -- Virtual package build string (e.g., `x86_64`).

**name: str**

**version: str | None**

**build: str | None**

**to_virtual_package()** → *conda.models.records.PackageRecord*

**class CondaSolver**

Bases: `NamedTuple`

Return type to use when defining a conda solver plugin hook.

For details on how this is used, see *conda_solvers()*.

**Parameters**

- **name** -- Solver name (e.g., `custom-solver`).

- **backend** -- Type that will be instantiated as the solver backend.

**name:** str

**backend:** type[*conda.core.solve.Solver*]

**class CondaPreCommand**

Bases: NamedTuple

Return type to use when defining a conda pre-command plugin hook.

For details on how this is used, see *conda_pre_commands()*.

**Parameters**

- **name** -- Pre-command name (e.g., `custom_plugin_pre_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

**name:** str

**action:** Callable[[str], None]

**run_for:** set[str]

**class CondaPostCommand**

Bases: NamedTuple

Return type to use when defining a conda post-command plugin hook.

For details on how this is used, see *conda_post_commands()*.

**Parameters**

- **name** -- Post-command name (e.g., `custom_plugin_post_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

**name:** str

**action:** Callable[[str], None]

**run_for:** set[str]

**class ChannelNameMixin**(*channel_name: str*, **args*, ***kwargs*)

Class mixin to make all plugin implementations compatible, e.g. when they use an existing (e.g. 3rd party) requests authentication handler.

Please use the concrete *ChannelAuthBase* in case you're creating an own implementation.

**class ChannelAuthBase**(*channel_name: str*, **args*, ***kwargs*)

Bases: *ChannelNameMixin*, `requests.auth.AuthBase`

Base class that we require all plugin implementations to use to be compatible.

Authentication is tightly coupled with individual channels. Therefore, an additional `channel_name` property must be set on the `requests.auth.AuthBase` based class.

**class CondaAuthHandler**

Bases: NamedTuple

Return type to use when the defining the conda auth handlers hook.

**Parameters**

- **name** -- Name (e.g., `basic-auth`). This name should be unique and only one may be registered at a time.
- **handler** -- Type that will be used as the authentication handler during network requests.

**name:** str

**handler:** type[*ChannelAuthBase*]

**class CondaHealthCheck**

Bases: NamedTuple

Return type to use when defining conda health checks plugin hook.

**name:** str

**action:** Callable[[str, bool], None]

**class CondaPreSolve**

Return type to use when defining a conda pre-solve plugin hook.

For details on how this is used, see *conda_pre_solves()*.

**Parameters**

- **name** -- Pre-solve name (e.g., `custom_plugin_pre_solve`).
- **action** -- Callable which contains the code to be run.

**name:** str

**action:** Callable[[frozenset[*conda.models.match_spec.MatchSpec*], frozenset[*conda.models.match_spec.MatchSpec*]], None]

**class CondaPostSolve**

Return type to use when defining a conda post-solve plugin hook.

For details on how this is used, see *conda_post_solves()*.

**Parameters**

- **name** -- Post-solve name (e.g., `custom_plugin_post_solve`).
- **action** -- Callable which contains the code to be run.

**name:** str

**action:** Callable[[str, tuple[*conda.models.records.PackageRecord*, Ellipsis], tuple[*conda.models.records.PackageRecord*, Ellipsis]], None]

**class CondaSetting**

Return type to use when defining a conda setting plugin hook.

For details on how this is used, see *conda_settings()*.

**Parameters**

- **name** -- name of the setting (e.g., `config_param`)

- **description** -- description of the setting that should be targeted towards users of the plugin
- **parameter** -- Parameter instance containing the setting definition
- **aliases** -- alternative names of the setting

**name:** *str*

**description:** *str*

**parameter:** *conda.common.configuration.Parameter*

**aliases:** *tuple[str, Ellipsis]*

**class ProgressBarBase**(*description: str*, ***kwargs*)

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

**abstract update_to**(*fraction*) → None

**abstract refresh**() → None

**abstract close**() → None

**finish**()

**classmethod get_lock**()

**class SpinnerBase**(*message: str*, *fail_message: str = 'failed\n'*)

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

**abstract __enter__**()

**abstract __exit__**(*exc_type*, *exc_val*, *exc_tb*)

**class ReporterRendererBase**

Bases: abc.ABC

Base class for all reporter renderers.

**render**(*data: Any*, ***kwargs*) → str

**abstract detail_view**(*data: dict[str, str | int | bool]*, ***kwargs*) → str

Render the output in a "tabular" format.

**abstract envs_list**(*data*, ***kwargs*) → str

Render a list of environments

**abstract progress_bar**(*description: str*, ***kwargs*) → *ProgressBarBase*

Return a ProgressBarBase~ object to use as a progress bar

**classmethod progress_bar_context_manager**() → contextlib.AbstractContextManager

Returns a null context by default but allows plugins to define their own if necessary

**abstract spinner**(*message*, *failed_message*) → *SpinnerBase*

Return a SpinnerBase~ object to use as a spinner (i.e. loading dialog)

**abstract prompt**(*message: str = 'Proceed'*, *choices=('yes', 'no')*, *default: str = 'yes'*) → str

Allows for defining an implementation of a "yes/no" confirmation function

**class CondaReporterBackend**

Return type to use when defining a conda reporter backend plugin hook.

For details on how this is used, see: *conda_reporter_backends()*.

**Parameters**

- **name** -- name of the reporter backend (e.g., `email_reporter`) This is how the reporter backend with be references in configuration files.
- **description** -- short description of what the reporter handler does
- **renderer** -- implementation of `ReporterRendererBase` that will be used as the reporter renderer

**name:** str

**description:** str

**renderer:** type[*ReporterRendererBase*]

**class CondaRequestHeader**

Define vendor specific headers to include HTTP requests

For details on how this is used, see *conda_request_headers()* and *conda_session_headers()*.

**Parameters**

- **name** -- name of the header used in the HTTP request
- **value** -- value of the header used in the HTTP request

**name:** str

**value:** str

**class CondaPreTransactionAction**

Return type to use when defining a pre-transaction action hook.

For details on how this is used, see *conda_pre_transaction_actions()*.

**Parameters**

- **name** -- Pre transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**name:** str

**action:** type[*conda.core.path_actions.Action*]

**class CondaPostTransactionAction**

Return type to use when defining a post-transaction action hook.

For details on how this is used, see *conda_post_transaction_actions()*.

**Parameters**

- **name** -- Post transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**name:** str

**action:** type[*conda.core.path_actions.Action*]

class CondaPrefixDataLoader

Define new loaders to expose non-conda packages in a given prefix as PrefixRecord objects.

**Parameters**

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to PrefixRecord objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

**name:** str

**loader:** CondaPrefixDataLoaderCallable

class EnvironmentSpecBase

Bases: abc.ABC

**EXPERIMENTAL**

Base class for all env specs.

**abstract property env:** *conda.models.environment.Environment*

Express the provided environment file as a conda environment object.

**Returns Environment**
the conda environment represented by the file.

**detection_supported:** ClassVar[bool] = True

**abstract can_handle**() → bool

Determines if the EnvSpec plugin can read and operate on the environment described by the *filename*.

**Returns bool**
returns True, if the plugin can interpret the file.

class CondaEnvironmentSpecifier

**EXPERIMENTAL**

Return type to use when defining a conda env spec plugin hook.

For details on how this is used, see *conda_environment_specifiers()*.

**Parameters**

- **name** -- name of the spec (e.g., environment_yaml)
- **environment_spec** -- EnvironmentSpecBase subclass handler

**name:** str

**environment_spec:** type[*EnvironmentSpecBase*]

### virtual_packages

### archspec

Detect archspec name.

## Functions

| | |
|---|---|
| *conda_virtual_packages*() | |

**conda_virtual_packages**()

### conda

Expose conda version.

## Functions

| | |
|---|---|
| *conda_virtual_packages*() | |

**conda_virtual_packages**()

### cuda

Detect CUDA version.

## Functions

| | |
|---|---|
| *cuda_version*() | Attempt to detect the version of CUDA present in the operating system. |
| *cached_cuda_version*() | A cached version of the cuda detection system. |
| *conda_virtual_packages*() | |
| *_cuda_driver_version_detector_target*(queue) | Attempt to detect the version of CUDA present in the operating system in a |

**cuda_version**()

Attempt to detect the version of CUDA present in the operating system.

On Windows and Linux, the CUDA library is installed by the NVIDIA driver package, and is typically found in the standard library path, rather than with the CUDA SDK (which is optional for running CUDA apps).

On macOS, the CUDA library is only installed with the CUDA SDK, and might not be in the library path.

Returns: version string (e.g., '9.2') or None if CUDA is not found.

`cached_cuda_version()`

A cached version of the cuda detection system.

`conda_virtual_packages()`

`_cuda_driver_version_detector_target(queue)`

Attempt to detect the version of CUDA present in the operating system in a subprocess.

On Windows and Linux, the CUDA library is installed by the NVIDIA driver package, and is typically found in the standard library path, rather than with the CUDA SDK (which is optional for running CUDA apps).

On macOS, the CUDA library is only installed with the CUDA SDK, and might not be in the library path.

**Returns: version string (e.g., '9.2') or None if CUDA is not found.**
The result is put in the queue rather than a return value.

### freebsd

Detect whether this is FeeBSD.

## Functions

*conda_virtual_packages*()

`conda_virtual_packages()`

### linux

Detect whether this is Linux.

## Functions

*conda_virtual_packages*()

`conda_virtual_packages()`

#### osx

Detect whether this is macOS.

### Functions

| |
|---|
| *conda_virtual_packages*() |

**conda_virtual_packages**()

#### windows

Detect whether this is Windows.

### Functions

| |
|---|
| *conda_virtual_packages*() |

**conda_virtual_packages**()

**plugins**

### Classes

| | |
|---|---|
| *CondaAuthHandler* | Return type to use when the defining the conda auth handlers hook. |
| *CondaEnvironmentSpecifier* | **EXPERIMENTAL** |
| *CondaHealthCheck* | Return type to use when defining conda health checks plugin hook. |
| *CondaPostCommand* | Return type to use when defining a conda post-command plugin hook. |
| *CondaPostSolve* | Return type to use when defining a conda post-solve plugin hook. |
| *CondaPostTransactionAction* | Return type to use when defining a post-transaction action hook. |
| *CondaPreCommand* | Return type to use when defining a conda pre-command plugin hook. |
| *CondaPrefixDataLoader* | Define new loaders to expose non-conda packages in a given prefix |
| *CondaPreSolve* | Return type to use when defining a conda pre-solve plugin hook. |
| *CondaPreTransactionAction* | Return type to use when defining a pre-transaction action hook. |
| *CondaReporterBackend* | Return type to use when defining a conda reporter backend plugin hook. |
| *CondaRequestHeader* | Define vendor specific headers to include HTTP requests |
| *CondaSetting* | Return type to use when defining a conda setting plugin hook. |
| *CondaSolver* | Return type to use when defining a conda solver plugin hook. |
| *CondaSubcommand* | Return type to use when defining a conda subcommand plugin hook. |
| *CondaVirtualPackage* | Return type to use when defining a conda virtual package plugin hook. |

### Attributes

| | |
|---|---|
| *hookimpl* | Decorator used to mark plugin hook implementations |

**hookimpl**

Decorator used to mark plugin hook implementations

**class CondaAuthHandler**

Bases: `NamedTuple`

Return type to use when the defining the conda auth handlers hook.

**Parameters**

- **name** -- Name (e.g., `basic-auth`). This name should be unique and only one may be registered at a time.
- **handler** -- Type that will be used as the authentication handler during network requests.

**name:** str

**handler:** type[*ChannelAuthBase*]

class CondaEnvironmentSpecifier

**EXPERIMENTAL**

Return type to use when defining a conda env spec plugin hook.

For details on how this is used, see *conda_environment_specifiers()*.

**Parameters**

- **name** -- name of the spec (e.g., `environment_yaml`)
- **environment_spec** -- EnvironmentSpecBase subclass handler

**name:** str

**environment_spec:** type[*EnvironmentSpecBase*]

class CondaHealthCheck

Bases: `NamedTuple`

Return type to use when defining conda health checks plugin hook.

**name:** str

**action:** Callable[[str, bool], None]

class CondaPostCommand

Bases: `NamedTuple`

Return type to use when defining a conda post-command plugin hook.

For details on how this is used, see *conda_post_commands()*.

**Parameters**

- **name** -- Post-command name (e.g., `custom_plugin_post_commands`).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. `install` or `create`).

**name:** str

**action:** Callable[[str], None]

**run_for:** set[str]

class CondaPostSolve

Return type to use when defining a conda post-solve plugin hook.

For details on how this is used, see *conda_post_solves()*.

**Parameters**

- **name** -- Post-solve name (e.g., `custom_plugin_post_solve`).
- **action** -- Callable which contains the code to be run.

**name:** str

**action:** Callable[[str, tuple[*conda.models.records.PackageRecord*, Ellipsis], tuple[*conda.models.records.PackageRecord*, Ellipsis]], None]

**class CondaPostTransactionAction**

Return type to use when defining a post-transaction action hook.

For details on how this is used, see *conda_post_transaction_actions()*.

**Parameters**

- **name** -- Post transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**name:** str

**action:** type[*conda.core.path_actions.Action*]

**class CondaPreCommand**

Bases: NamedTuple

Return type to use when defining a conda pre-command plugin hook.

For details on how this is used, see *conda_pre_commands()*.

**Parameters**

- **name** -- Pre-command name (e.g., custom_plugin_pre_commands).
- **action** -- Callable which contains the code to be run.
- **run_for** -- Represents the command(s) this will be run on (e.g. install or create).

**name:** str

**action:** Callable[[str], None]

**run_for:** set[str]

**class CondaPrefixDataLoader**

Define new loaders to expose non-conda packages in a given prefix as PrefixRecord objects.

**Parameters**

- **name** -- name of the loader
- **loader** -- a function that takes a prefix and a dictionary that maps package names to PrefixRecord objects. The newly loaded packages must be inserted in the passed dictionary accordingly, and also returned as a separate dictionary.

**name:** str

**loader:** CondaPrefixDataLoaderCallable

**class CondaPreSolve**

Return type to use when defining a conda pre-solve plugin hook.

For details on how this is used, see *conda_pre_solves()*.

**Parameters**

- **name** -- Pre-solve name (e.g., custom_plugin_pre_solve).
- **action** -- Callable which contains the code to be run.

**name:** *str*

**action:** *Callable[[frozenset[conda.models.match_spec.MatchSpec], frozenset[conda.models.match_spec.MatchSpec]], None]*

### class CondaPreTransactionAction

Return type to use when defining a pre-transaction action hook.

For details on how this is used, see *conda_pre_transaction_actions()*.

**Parameters**

- **name** -- Pre transaction name (this is just a label)
- **action** -- Action class which implements plugin behavior. See *Action* for implementation details

**name:** *str*

**action:** *type[conda.core.path_actions.Action]*

### class CondaReporterBackend

Return type to use when defining a conda reporter backend plugin hook.

For details on how this is used, see: *conda_reporter_backends()*.

**Parameters**

- **name** -- name of the reporter backend (e.g., `email_reporter`) This is how the reporter backend with be references in configuration files.
- **description** -- short description of what the reporter handler does
- **renderer** -- implementation of `ReporterRendererBase` that will be used as the reporter renderer

**name:** *str*

**description:** *str*

**renderer:** *type[ReporterRendererBase]*

### class CondaRequestHeader

Define vendor specific headers to include HTTP requests

For details on how this is used, see *conda_request_headers()* and *conda_session_headers()*.

**Parameters**

- **name** -- name of the header used in the HTTP request
- **value** -- value of the header used in the HTTP request

**name:** *str*

**value:** *str*

### class CondaSetting

Return type to use when defining a conda setting plugin hook.

For details on how this is used, see *conda_settings()*.

**Parameters**

- **name** -- name of the setting (e.g., `config_param`)

- **description** -- description of the setting that should be targeted towards users of the plugin
- **parameter** -- Parameter instance containing the setting definition
- **aliases** -- alternative names of the setting

**name:** str

**description:** str

**parameter:** *conda.common.configuration.Parameter*

**aliases:** tuple[str, Ellipsis]

class CondaSolver

Bases: NamedTuple

Return type to use when defining a conda solver plugin hook.

For details on how this is used, see *conda_solvers()*.

**Parameters**

- **name** -- Solver name (e.g., `custom-solver`).
- **backend** -- Type that will be instantiated as the solver backend.

**name:** str

**backend:** type[*conda.core.solve.Solver*]

class CondaSubcommand

Return type to use when defining a conda subcommand plugin hook.

For details on how this is used, see *conda_subcommands()*.

**Parameters**

- **name** -- Subcommand name (e.g., `conda my-subcommand-name`).
- **summary** -- Subcommand summary, will be shown in `conda --help`.
- **action** -- Callable that will be run when the subcommand is invoked.
- **configure_parser** -- Callable that will be run when the subcommand parser is initialized.

**name:** str

**summary:** str

**action:** Callable[[argparse.Namespace | tuple[str]], int | None]

**configure_parser:** Callable[[argparse.ArgumentParser], None] | None

class CondaVirtualPackage

Bases: NamedTuple

Return type to use when defining a conda virtual package plugin hook.

For details on how this is used, see *conda_virtual_packages()*.

**Parameters**

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).

- **build** -- Virtual package build string (e.g., `x86_64`).

**name:** str

**version:** str | None

**build:** str | None

**to_virtual_package**() → *conda.models.records.PackageRecord*

## reporters

Holds functions for output rendering in conda

### Functions

| | |
|---|---|
| *_get_render_func*(→ Callable) | Retrieves the render function to use |
| *render*(→ None) | Used to render output in conda |
| *get_progress_bar*(→ conda.plugins.types.ProgressBarBase) | Retrieve the progress bar for the currently configured reporter backend |
| *get_progress_bar_context_manager*(...) | Retrieve progress bar context manager to use with registered reporter |
| *get_spinner*(→ conda.plugins.types.SpinnerBase) | Retrieve spinner to use with registered reporter |
| *confirm_yn*(→ bool) | Display a "yes/no" confirmation input |

### Attributes

| |
|---|
| *logger* |

**logger**

**_get_render_func**(*style: str | None = None*) → Callable

Retrieves the render function to use

**render**(*data*, *style: str | None = None*, ***kwargs*) → None

Used to render output in conda

The output will either be rendered as "json" or normal "console" output to stdout. This function allows us to configure different reporter backends for these two types of output.

**get_progress_bar**(*description: str*, ***kwargs*) → *conda.plugins.types.ProgressBarBase*

Retrieve the progress bar for the currently configured reporter backend

**get_progress_bar_context_manager**() → contextlib.AbstractContextManager

Retrieve progress bar context manager to use with registered reporter

**get_spinner**(*message: str*, *fail_message: str = 'failed\n'*) → *conda.plugins.types.SpinnerBase*

Retrieve spinner to use with registered reporter

**confirm_yn**(*message: str = 'Proceed'*, *default='yes'*, *dry_run=None*) → bool

Display a "yes/no" confirmation input

### resolve

Low-level SAT solver wrapper/interface for the classic solver.

See conda.core.solver.Solver for the high-level API.

## Classes

| | |
|---|---|
| *Resolve* | |

## Functions

| | |
|---|---|
| *_get_sat_solver_cls*([sat_solver_choice]) | |
| *exactness_and_number_of_deps*(resolve_obj, ms) | Sorting key to emphasize packages that have more strict |

## Attributes

| | |
|---|---|
| *stdoutlog* | |
| *Unsatisfiable* | |
| *ResolvePackageNotFound* | |
| *_sat_solvers* | |

**stdoutlog**

**Unsatisfiable**

**ResolvePackageNotFound**

**_sat_solvers**

**_get_sat_solver_cls**(*sat_solver_choice=SatSolverChoice.PYCOSAT*)

**exactness_and_number_of_deps**(*resolve_obj*, *ms*)

Sorting key to emphasize packages that have more strict requirements. More strict means the reduced index can be reduced more, so we want to consider these more constrained deps earlier in reducing the index.

**class Resolve**(*index*, *processed=False*, *channels=()*)

**__hash__**()

Return hash(self).

**default_filter**(*features=None*, *filter=None*)

`valid`(*spec_or_prec*, *filter*, *optional=True*)

Tests if a package, MatchSpec, or a list of both has satisfiable dependencies, assuming cyclic dependencies are always valid.

**Parameters**

- `spec_or_prec` -- a package record, a MatchSpec, or an iterable of these.
- `filter` -- a dictionary of (fkey,valid) pairs, used to consider a subset of dependencies, and to eliminate repeated searches.
- `optional` -- if True (default), do not enforce optional specifications when considering validity. If False, enforce them.

**Returns**

True if the full set of dependencies can be satisfied; False otherwise. If filter is supplied and update is True, it will be updated with the search results.

`valid2`(*spec_or_prec*, *filter_out*, *optional=True*)

`invalid_chains`(*spec*, *filter*, *optional=True*)

Constructs a set of 'dependency chains' for invalid specs.

A dependency chain is a tuple of MatchSpec objects, starting with the requested spec, proceeding down the dependency tree, ending at a specification that cannot be satisfied.

**Parameters**

- `spec` -- a package key or MatchSpec
- `filter` -- a dictionary of (prec, valid) pairs to be used when testing for package validity.

**Returns**

A tuple of tuples, empty if the MatchSpec is valid.

`verify_specs`(*specs*)

Perform a quick verification that specs and dependencies are reasonable.

**Parameters**

`specs` -- An iterable of strings or MatchSpec objects to be tested.

**Returns**

Nothing, but if there is a conflict, an error is thrown.

Note that this does not attempt to resolve circular dependencies.

`_classify_bad_deps`(*bad_deps*, *specs_to_add*, *history_specs*, *strict_channel_priority*)

`find_matches_with_strict`(*ms*, *strict_channel_priority*)

`find_conflicts`(*specs*, *specs_to_add=None*, *history_specs=None*)

`breadth_first_search_for_dep_graph`(*root_spec*, *target_name*, *dep_graph*, *num_targets=1*)

Return shorted path from root_spec to target_name

`build_graph_of_deps`(*spec*)

`build_conflict_map`(*specs*, *specs_to_add=None*, *history_specs=None*)

Perform a deeper analysis on conflicting specifications, by attempting to find the common dependencies that might be the cause of conflicts.

**Parameters**

- `specs` -- An iterable of strings or MatchSpec objects to be tested.

- **conflict.** (*It is assumed that the specs*)

**Returns**
A list of lists of bad deps

**Return type**
bad_deps

**Strategy:**
If we're here, we know that the specs conflict. This could be because: - One spec conflicts with another; e.g.

['numpy 1.5*', 'numpy >=1.6']

- **One spec conflicts with a dependency of another; e.g.**
  ['numpy 1.5*', 'scipy 0.12.0b1']
- **Each spec depends on *the same package* but in a different way; e.g.,**
  ['A', 'B'] where A depends on numpy 1.5, and B on numpy 1.6.

Technically, all three of these cases can be boiled down to the last one if we treat the spec itself as one of the "dependencies". There might be more complex reasons for a conflict, but this code only considers the ones above.

The purpose of this code, then, is to identify packages (like numpy above) that all of the specs depend on *but in different ways*. We then identify the dependency chains that lead to those packages.

**_get_strict_channel**(*package_name*)

**_broader**(*ms*, *specs_by_name*)

Prevent introduction of matchspecs that broaden our selection of choices.

**_get_package_pool**(*specs*)

**get_reduced_index**(*explicit_specs*, *sort_by_exactness=True*, *exit_on_conflict=False*)

**match_any**(*mss*, *prec*)

**find_matches**(*spec:* conda.models.match_spec.MatchSpec) → tuple[*conda.models.records.PackageRecord*]

**ms_depends**(*prec:* conda.models.records.PackageRecord) → list[*conda.models.match_spec.MatchSpec*]

**version_key**(*prec*, *vtype=None*)

**static _make_channel_priorities**(*channels*)

**get_pkgs**(*ms*, *emptyok=False*)

**static to_sat_name**(*val*)

**static to_feature_metric_id**(*prec_dist_str*, *feat*)

**push_MatchSpec**(*C*, *spec*)

**gen_clauses**()

**generate_spec_constraints**(*C*, *specs*)

**generate_feature_count**(*C*)

**generate_update_count**(*C*, *specs*)

**generate_feature_metric**(*C*)

**generate_removal_count**(*C*, *specs*)

**generate_install_count**(*C*, *specs*)

**generate_package_count**(*C*, *missing*)

**generate_version_metrics**(*C*, *specs*, *include0=False*)

**dependency_sort**(*must_have: dict[str,* conda.models.records.PackageRecord*]*) → list[*conda.models.records.PackageRecord*]

**environment_is_consistent**(*installed*)

**get_conflicting_specs**(*specs*, *explicit_specs*)

**bad_installed**(*installed*, *new_specs*)

**restore_bad**(*pkgs*, *preserve*)

**install_specs**(*specs*, *installed*, *update_deps=True*)

**install**(*specs*, *installed=None*, *update_deps=True*, *returnall=False*)

**remove_specs**(*specs*, *installed*)

**remove**(*specs*, *installed*)

**solve**(*specs: list*, *returnall: bool = False*, *_remove=False*, *specs_to_add=None*, *history_specs=None*, *should_retry_solve=False*) → list[*conda.models.records.PackageRecord*]

### testing

### cases

Extends unittest.TestCase to include select pytest fixtures.

## Classes

| | |
|---|---|
| *BaseTestCase* | A class whose instances are single test cases. |

**class BaseTestCase**(*methodName='runTest'*)

Bases: unittest.TestCase

A class whose instances are single test cases.

By default, the test code itself should be placed in a method named 'runTest'.

If the fixture may be used for many test cases, create as many test methods as are needed. When instantiating such a TestCase subclass, specify in the constructor arguments the name of the test method that the instance is to execute.

Test authors should subclass TestCase for their own tests. Construction and deconstruction of the test's environment ('fixture') can be implemented by overriding the 'setUp' and 'tearDown' methods respectively.

If it is necessary to override the __init__ method, the base class __init__ method must always be called. It is important that subclasses should not change the signature of their __init__ method, since instances of the classes are instantiated automatically by parts of the framework in order to be run.

When subclassing TestCase, you can set these attributes: * failureException: determines which exception will be raised when

the instance's assertion methods fail; test methods raising this exception will be deemed to have 'failed' rather than 'errored'.

- **longMessage: determines whether long messages (including repr of**
  objects used in assert methods) will be printed on failure in *addition* to any explicit message passed.
- **maxDiff: sets the maximum length of a diff in failure messages**
  by assert methods using difflib. It is looked up as an instance attribute so can be configured by individual tests if required.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

**fixture_names = ('tmpdir',)**

**auto_injector_fixture**(*request*)

### fixtures

Collection of pytest fixtures used in conda tests.

## Classes

| |
|---|
| *CondaCLIFixture* |
| *PathFactoryFixture* |
| *TmpEnvFixture* |
| *TmpChannelFixture* |

### Functions

| | |
|---|---|
| *suppress_resource_warning*() | Suppress *Unclosed Socket Warning* |
| *tmpdir*(tmpdir, request) | |
| *clear_subdir_cache*() | |
| *reset_conda_context*() | Resets the context object after each test function is run. |
| *temp_package_cache*(tmp_path_factory) | Used to isolate package or index cache from other tests. |
| *parametrized_solver_fixture*(...) | A parameterized fixture that sets the solver backend to (1) libmamba |
| *solver_classic*(...) | |
| *solver_libmamba*(...) | |
| *_solver_helper*(→ collections.abc.Iterable[Solver]) | |
| *session_capsys*(...) | |
| *conda_cli*(→ collections.abc.Iterator[CondaCLIFixture]) | A function scoped fixture returning CondaCLIFixture instance. |
| *session_conda_cli*(...) | A session scoped fixture returning CondaCLIFixture instance. |
| *path_factory*(...) | A function scoped fixture returning PathFactoryFixture instance. |
| *tmp_env*(→ collections.abc.Iterator[TmpEnvFixture]) | A function scoped fixture returning TmpEnvFixture instance. |
| *session_tmp_env*(→ collections.abc.Iterator[TmpEnvFixture]) | A session scoped fixture returning TmpEnvFixture instance. |
| *tmp_channel*(→ collections.abc.Iterator[TmpChannelFixture]) | A function scoped fixture returning TmpChannelFixture instance. |
| *context_aware_monkeypatch*(→ pytest.MonkeyPatch) | A monkeypatch fixture that resets context after each test. |
| *tmp_pkgs_dir*(→ collections.abc.Iterator[pathlib.Path]) | A function scoped fixture returning a temporary package cache directory. |
| *tmp_envs_dir*(→ collections.abc.Iterator[pathlib.Path]) | A function scoped fixture returning a temporary environment directory. |
| *PYTHONPATH*() | We need to set this so Python loads the dev version of 'conda', usually taken |

### Attributes

| |
|---|
| *Solver* |

**suppress_resource_warning**()

Suppress *Unclosed Socket Warning*

It seems urllib3 keeps a socket open to avoid costly recreation costs.

xref: https://github.com/kennethreitz/requests/issues/1882

**tmpdir**(*tmpdir*, *request*)

**clear_subdir_cache**()

**reset_conda_context**()

Resets the context object after each test function is run.

**temp_package_cache**(*tmp_path_factory*)

Used to isolate package or index cache from other tests.

**parametrized_solver_fixture**(*request: pytest.FixtureRequest*, *monkeypatch: pytest.MonkeyPatch*) → collections.abc.Iterable[Literal[libmamba, classic]]

A parameterized fixture that sets the solver backend to (1) libmamba and (2) classic for each test. It's using autouse=True, so only import it in modules that actually need it.

Note that skips and xfails need to be done _inside_ the test body. Decorators can't be used because they are evaluated before the fixture has done its work!

So, instead of:

@pytest.mark.skipif(context.solver == "libmamba", reason="...") def test_foo():

...

Do:

**def test_foo():**

**if context.solver == "libmamba":**
pytest.skip("...")

...

**solver_classic**(*request: pytest.FixtureRequest*, *monkeypatch: pytest.MonkeyPatch*) → collections.abc.Iterable[Literal[classic]]

**solver_libmamba**(*request: pytest.FixtureRequest*, *monkeypatch: pytest.MonkeyPatch*) → collections.abc.Iterable[Literal[libmamba]]

**Solver**

**_solver_helper**(*request: pytest.FixtureRequest*, *monkeypatch: pytest.MonkeyPatch*, *solver:* Solver) → collections.abc.Iterable[*Solver*]

**session_capsys**(*request*) → collections.abc.Iterator[_pytest.capture.MultiCapture]

**class CondaCLIFixture**

**capsys: pytest.CaptureFixture | None**

**__call__**(**argv: str | os.PathLike[str] | pathlib.Path*, *raises: type[Exception] | tuple[type[Exception], Ellipsis]*) → tuple[str, str, pytest.ExceptionInfo]

**__call__**(**argv: str | os.PathLike[str] | pathlib.Path*) → tuple[str, str, int]

Test conda CLI. Mimic what is done in *conda.cli.main.main*.

*conda ... == conda_cli(...)*

**Parameters**

- **argv** -- Arguments to parse.

- **raises** -- Expected exception to intercept. If provided, the raised exception will be returned instead of exit code (see pytest.raises and pytest.ExceptionInfo).

**Returns**
Command results (stdout, stderr, exit code or pytest.ExceptionInfo).

**static _cast_args**(*argv: tuple[str | os.PathLike[str] | pathlib.Path, Ellipsis]*) → collections.abc.Iterable[str]

Cast args to string and inspect for *conda run*.

*conda run* is a unique case that requires *--dev* to use the src shell scripts and not the shell scripts provided by the installer.

**conda_cli**(*capsys: pytest.CaptureFixture*) → collections.abc.Iterator[*CondaCLIFixture*]

A function scoped fixture returning CondaCLIFixture instance.

Use this for any commands that are local to the current test (e.g., creating a conda environment only used in the test).

**session_conda_cli**() → collections.abc.Iterator[*CondaCLIFixture*]

A session scoped fixture returning CondaCLIFixture instance.

Use this for any commands that are global to the test session (e.g., creating a conda environment shared across tests, *conda info*, etc.).

**class PathFactoryFixture**

**tmp_path:** pathlib.Path

**__call__**(*name: str | None = None*, *prefix: str | None = None*, *suffix: str | None = None*) → pathlib.Path

Unique, non-existent path factory.

Extends pytest's *tmp_path* fixture with a new unique, non-existent path for usage in cases where we need a temporary path that doesn't exist yet.

**Parameters**

- **name** -- Path name to append to *tmp_path*
- **prefix** -- Prefix to prepend to unique name generated
- **suffix** -- Suffix to append to unique name generated

**Returns**
A new unique path

**path_factory**(*tmp_path: pathlib.Path*) → collections.abc.Iterator[*PathFactoryFixture*]

A function scoped fixture returning PathFactoryFixture instance.

Use this to generate any number of temporary paths for the test that are unique and do not exist yet.

**class TmpEnvFixture**

**path_factory:** *PathFactoryFixture* | pytest.TempPathFactory

**conda_cli:** *CondaCLIFixture*

**get_path**() → pathlib.Path

**__call__**(**packages: str*, *prefix: str | os.PathLike | None = None*) → collections.abc.Iterator[pathlib.Path]

Generate a conda environment with the provided packages.

**Parameters**

- **packages** -- The packages to install into environment
- **prefix** -- The prefix at which to install the conda environment

**Returns**

The conda environment's prefix

**tmp_env**(*path_factory: PathFactoryFixture*, *conda_cli: CondaCLIFixture*) → collections.abc.Iterator[*TmpEnvFixture*]

A function scoped fixture returning TmpEnvFixture instance.

Use this when creating a conda environment that is local to the current test.

**session_tmp_env**(*tmp_path_factory: pytest.TempPathFactory*, *session_conda_cli: CondaCLIFixture*) → collections.abc.Iterator[*TmpEnvFixture*]

A session scoped fixture returning TmpEnvFixture instance.

Use this when creating a conda environment that is shared across tests.

**class TmpChannelFixture**

**path_factory:** *PathFactoryFixture*

**conda_cli:** *CondaCLIFixture*

**__call__**(**specs: str*) → collections.abc.Iterator[tuple[pathlib.Path, str]]

**tmp_channel**(*path_factory: PathFactoryFixture*, *conda_cli: CondaCLIFixture*) → collections.abc.Iterator[*TmpChannelFixture*]

A function scoped fixture returning TmpChannelFixture instance.

**context_aware_monkeypatch**(*monkeypatch: pytest.MonkeyPatch*) → pytest.MonkeyPatch

A monkeypatch fixture that resets context after each test.

**tmp_pkgs_dir**(*path_factory: PathFactoryFixture*, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary package cache directory.

**tmp_envs_dir**(*path_factory: PathFactoryFixture*, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary environment directory.

**PYTHONPATH**()

We need to set this so Python loads the dev version of 'conda', usually taken from *conda/* in the root of the cloned repo. This root is usually the working directory when we run *pytest*. Otherwise, it will import the one installed in the base environment, which might have not been overwritten with *pip install -e . --no-deps*. This doesn't happen in other tests because they run with the equivalent of *python -m conda*. However, some tests directly run *conda (shell function) which calls `conda* (Python entry point). When a script is called this way, it bypasses the automatic "working directory is first on sys.path" behavior you find in *python -m* style calls. See https://docs.python.org/3/library/sys_path_init.html for details.

### gateways

### fixtures

Collection of pytest fixtures used in conda.gateways tests.

#### Functions

| | |
|---|---|
| *minio_s3_server*(xprocess, tmp_path) | Mock a local S3 server using *minio* |

#### Attributes

| |
|---|
| *MINIO_EXE* |

**MINIO_EXE**

**minio_s3_server**(*xprocess*, *tmp_path*)

Mock a local S3 server using *minio*

This requires: - pytest-xprocess: runs the background process - minio: the executable must be in PATH

Note, the given S3 server will be EMPTY! The test function needs to populate it. You can use *conda.testing.helpers.populate_s3_server* for that.

### helpers

Collection of helper functions used in conda tests.

#### Functions

| |
|---|
| *strip_expected*(stderr) |
| *raises*(exception, func[, string]) |
| *captured*([disallow_stderr]) |
| *assert_equals*(a, b[, output]) |
| *assert_not_in*(a, b[, output]) |
| *assert_in*(a, b[, output]) |
| *add_subdir*(dist_string) |

continues on next page

Table 3 – continued from previous page

| | |
|---|---|
| *add_subdir_to_iter*(iterable) | |
| *tempdir*() | |
| *supplement_index_with_repodata*(index, repodata, ...) | |
| *add_feature_records_legacy*(index) | |
| *_export_subdir_data_to_repodata*(subdir_data) | This function is only temporary and meant to patch wrong / undesirable |
| *_sync_channel_to_disk*(subdir_data) | This function is only temporary and meant to patch wrong / undesirable |
| *_alias_canonical_channel_name_cache_to_file_* | This function is only temporary and meant to patch wrong / undesirable |
| *_patch_for_local_exports*(name, subdir_data) | This function is only temporary and meant to patch wrong / undesirable |
| *_get_index_r_base*(json_filename_or_packages, channel_name) | |
| *get_index_r_1*([subdir, add_pip, merge_noarch]) | |
| *get_index_r_2*([subdir, add_pip, merge_noarch]) | |
| *get_index_r_4*([subdir, add_pip, merge_noarch]) | |
| *get_index_r_5*([subdir, add_pip, merge_noarch]) | |
| *get_index_must_unfreeze*([subdir, add_pip, merge_noarch]) | |
| *get_index_cuda*([subdir, add_pip, merge_noarch]) | |
| *record*([name, version, depends, build, build_number, ...]) | |
| *_get_solver_base*(channel_id, tmpdir[, specs_to_add, ...]) | |
| *get_solver*(tmpdir[, specs_to_add, specs_to_remove, ...]) | |
| *get_solver_2*(tmpdir[, specs_to_add, specs_to_remove, ...]) | |
| *get_solver_4*(tmpdir[, specs_to_add, specs_to_remove, ...]) | |
| *get_solver_5*(tmpdir[, specs_to_add, specs_to_remove, ...]) | |
| *get_solver_aggregate_1*(tmpdir[, specs_to_add, ...]) | |
| *get_solver_aggregate_2*(tmpdir[, specs_to_add, ...]) | |
| *get_solver_must_unfreeze*(tmpdir[, specs_to_add, ...]) | |
| *get_solver_cuda*(tmpdir[, specs_to_add, ...]) | |
| *convert_to_dist_str*(solution) | |

continues on next page

Table 3 – continued from previous page

| | |
|---|---|
| *solver_class*() | |
| *in_subprocess*() | |
| *forward_to_subprocess*(→ subprocess.CompletedProcess \| None) | |

### Attributes

| |
|---|
| *TEST_DATA_DIR* |
| *CHANNEL_DIR_V2* |
| *EXPORTED_CHANNELS_DIR* |
| *expected_error_prefix* |

**TEST_DATA_DIR**

**CHANNEL_DIR_V2**

**EXPORTED_CHANNELS_DIR**

**expected_error_prefix = 'Using Anaconda Cloud api site https://api.anaconda.org'**

**strip_expected**(*stderr*)

**raises**(*exception*, *func*, *string=None*)

**captured**(*disallow_stderr=True*)

**assert_equals**(*a*, *b*, *output=''*)

**assert_not_in**(*a*, *b*, *output=''*)

**assert_in**(*a*, *b*, *output=''*)

**add_subdir**(*dist_string*)

**add_subdir_to_iter**(*iterable*)

**tempdir**()

**supplement_index_with_repodata**(*index*, *repodata*, *channel*, *priority*)

**add_feature_records_legacy**(*index*)

**_export_subdir_data_to_repodata**(*subdir_data:* conda.core.subdir_data.SubdirData)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

**_sync_channel_to_disk**(*subdir_data:* conda.core.subdir_data.SubdirData)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

**_alias_canonical_channel_name_cache_to_file_prefixed**(*name*, *subdir_data=None*)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

**_patch_for_local_exports**(*name*, *subdir_data*)

This function is only temporary and meant to patch wrong / undesirable testing behaviour. It should end up being replaced with the new class-based, backend-agnostic solver tests.

**_get_index_r_base**(*json_filename_or_packages*, *channel_name*, *subdir=context.subdir*, *add_pip=False*, *merge_noarch=False*)

**get_index_r_1**(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

**get_index_r_2**(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

**get_index_r_4**(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

**get_index_r_5**(*subdir=context.subdir*, *add_pip=False*, *merge_noarch=False*)

**get_index_must_unfreeze**(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

**get_index_cuda**(*subdir=context.subdir*, *add_pip=True*, *merge_noarch=False*)

**record**(*name='a'*, *version='1.0'*, *depends=None*, *build='0'*, *build_number=0*, *timestamp=0*, *channel=None*, ***kwargs*)

**_get_solver_base**(*channel_id*, *tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_2**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_4**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_5**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_aggregate_1**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_aggregate_2**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_must_unfreeze**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**get_solver_cuda**(*tmpdir*, *specs_to_add=()*, *specs_to_remove=()*, *prefix_records=()*, *history_specs=()*, *add_pip=False*, *merge_noarch=False*)

**convert_to_dist_str**(*solution*)

**solver_class**()

**in_subprocess**()

**forward_to_subprocess**(*request*, **cli_args*, ***subprocess_kwargs*) → subprocess.CompletedProcess | None

### integration

These helpers were originally defined in tests/test_create.py, but were refactored here so downstream projects can benefit from them too.

#### Classes

| |
|---|
| *Commands* |

#### Functions

| |
|---|
| *escape_for_winpath*(p) |
| *package_is_installed*(...) |
| *get_shortcut_dir*([prefix_for_unix]) |

#### Attributes

| |
|---|
| *TEST_LOG_LEVEL* |
| *PYTHON_BINARY* |
| *UNICODE_CHARACTERS* |
| *UNICODE_CHARACTERS_RESTRICTED* |
| *which_or_where* |
| *cp_or_copy* |
| *env_or_set* |
| *SPACER_CHARACTER* |

TEST_LOG_LEVEL

PYTHON_BINARY

UNICODE_CHARACTERS = 'öáêñßôç'

UNICODE_CHARACTERS_RESTRICTED = 'abcdef'

which_or_where

cp_or_copy

env_or_set

SPACER_CHARACTER = ' '

escape_for_winpath(*p*)

class Commands

COMPARE = 'compare'

CONFIG = 'config'

CLEAN = 'clean'

CREATE = 'create'

INFO = 'info'

INSTALL = 'install'

LIST = 'list'

REMOVE = 'remove'

SEARCH = 'search'

UPDATE = 'update'

RUN = 'run'

package_is_installed(*prefix: str | os.PathLike | pathlib.Path*, *spec: str* | conda.models.match_spec.MatchSpec, *reload_records: bool = True*) → *conda.models.records.PrefixRecord* | None

get_shortcut_dir(*prefix_for_unix=sys.prefix*)

### notices

### fixtures

Collection of pytest fixtures used in conda.notices tests.

### Functions

| | |
|---|---|
| *notices_cache_dir*(tmpdir) | Fixture that creates the notices cache dir while also mocking |
| *notices_mock_fetch_get_session*() | |
| *conda_notices_args_n_parser*() | |

**notices_cache_dir**(*tmpdir*)

Fixture that creates the notices cache dir while also mocking out a call to user_cache_dir.

**notices_mock_fetch_get_session**()

**conda_notices_args_n_parser**()

## helpers

Collection of helper functions used in conda.notices tests.

### Classes

| | |
|---|---|
| *DummyArgs* | Dummy object that sets all kwargs as object properties. |
| *MockResponse* | |

### Functions

| | |
|---|---|
| *get_test_notices*(→ dict) | |
| *add_resp_to_mock*(→ None) | Adds any number of MockResponse to MagicMock object as side_effects |
| *create_notice_cache_files*(→ None) | Creates the cache files that we use in tests |
| *offset_cache_file_mtime*(→ None) | Allows for offsetting the mtime of the notices cache file. This is often |
| *notices_decorator_assert_message_in_stdout*(c ...) | Tests a run of notices decorator where we expect to see the messages |
| *get_notice_cache_filenames*(→ tuple[str]) | Returns the filenames of the cache files that will be searched for |

### Attributes

| |
|---|
| *DEFAULT_NOTICE_MESG* |

**DEFAULT_NOTICE_MESG = 'Here is an example message that will be displayed to users'**

**get_test_notices**(*messages: collections.abc.Sequence[str]*, *level: str | None = 'info'*, *created_at: datetime.datetime | None = None*, *expired_at: datetime.datetime | None = None*) → dict

**add_resp_to_mock**(*mock_session: unittest.mock.MagicMock*, *status_code: int*, *messages_json: dict*, *raise_exc: bool = False*) → None

Adds any number of MockResponse to MagicMock object as side_effects

**create_notice_cache_files**(*cache_dir: pathlib.Path*, *cache_files: collections.abc.Sequence[str]*, *messages_json_seq: collections.abc.Sequence[dict]*) → None

Creates the cache files that we use in tests

**offset_cache_file_mtime**(*mtime_offset*) → None

Allows for offsetting the mtime of the notices cache file. This is often used to mock an older creation time the cache file.

**class DummyArgs**(***kwargs*)

Dummy object that sets all kwargs as object properties.

**notices_decorator_assert_message_in_stdout**(*captured*, *messages: collections.abc.Sequence[str]*, *dummy_mesg: str | None = None*, *not_in: bool = False*)

Tests a run of notices decorator where we expect to see the messages print to stdout.

**class MockResponse**(*status_code*, *json_data*, *raise_exc=False*)

**json**()

**get_notice_cache_filenames**(*ctx: conda.base.context.Context*) → tuple[str]

Returns the filenames of the cache files that will be searched for

## solver_helpers

Helpers for testing the solver.

### Classes

| | |
|---|---|
| *SimpleEnvironment* | Helper environment object. |
| *SolverTests* | Tests for *conda.core.solve.Solver* implementations. |

### Functions

| | |
|---|---|
| *index_packages*(num) | Get the index data of the `helpers.get_index_r_*` helpers. |
| *package_string*(record) | |
| *package_string_set*(packages) | Transforms package container in package string set. |
| *package_dict*(packages) | Transforms package container into a dictionary. |
| *empty_prefix*() | |
| *temp_simple_env*(→ SimpleEnvironment) | |

**index_packages**(*num*)

Get the index data of the `helpers.get_index_r_*` helpers.

**package_string**(*record*)

**package_string_set**(*packages*)

Transforms package container in package string set.

**package_dict**(*packages*)

Transforms package container into a dictionary.

**class SimpleEnvironment**(*path*, *solver_class*, *subdirs=context.subdirs*)

Helper environment object.

**property _channel_packages**

Helper that unfolds the `repo_packages` into a dictionary.

**REPO_DATA_KEYS = ('build', 'build_number', 'depends', 'license', 'md5', 'name', 'sha256', 'size', 'subdir',...**

**solver**(*add*, *remove*)

Writes `repo_packages` to the disk and creates a solver instance.

**solver_transaction**(*add=()*, *remove=()*, *as_specs=False*)

**install**(**specs*, *as_specs=False*)

**remove**(**specs*, *as_specs=False*)

**_package_data**(*record*)

Turn record into data, to be written in the JSON environment/repo files.

**_write_installed_packages**()

**_write_repo_packages**(*channel_name*, *packages*)

Write packages to the channel path.

**empty_prefix**()

**temp_simple_env**(*solver_class=Solver*) → *SimpleEnvironment*

**class SolverTests**

Tests for *conda.core.solve.Solver* implementations.

**abstract property solver_class:** type[*conda.core.solve.Solver*]

Class under test.

**property tests_to_skip**

**skip_tests**(*request*)

**env**()

**find_package_in_list**(*packages*, ***kwargs*)

**find_package**(***kwargs*)

**assert_unsatisfiable**(*exc_info*, *entries*)

Helper to assert that a *conda.exceptions.UnsatisfiableError* instance as a the specified set of unsatisfiable specifications.

**test_empty**(*env*)

**test_iopro_mkl**(*env*)

**test_iopro_nomkl**(*env*)

**test_mkl**(*env*)

**test_accelerate**(*env*)

**test_scipy_mkl**(*env*)

**test_anaconda_nomkl**(*env*)

**test_pseudo_boolean**(*env*)

**test_unsat_from_r1**(*env*)

**test_unsat_simple**(*env*)

**test_get_dists**(*env*)

**test_unsat_shortest_chain_1**(*env*)

**test_unsat_shortest_chain_2**(*env*)

**test_unsat_shortest_chain_3**(*env*)

**test_unsat_shortest_chain_4**(*env*)

**test_unsat_chain**(*env*)

**test_unsat_any_two_not_three**(*env*)

**test_unsat_expand_single**(*env*)

**test_unsat_missing_dep**(*env*)

**test_nonexistent**(*env*)

**test_timestamps_and_deps**(*env*)

**test_nonexistent_deps**(*env*)

**test_install_package_with_feature**(*env*)

**test_unintentional_feature_downgrade**(*env*)

**test_circular_dependencies**(*env*)

**test_irrational_version**(*env*)

**test_no_features**(*env*)

**test_channel_priority_1**(*monkeypatch*, *env*)

**test_unsat_channel_priority**(*monkeypatch*, *env*)

**test_remove**(*env*)

**test_surplus_features_1**(*env*)

**test_surplus_features_2**(*env*)

**test_get_reduced_index_broadening_with_unsatisfiable_early_dep**(*env*)

**test_get_reduced_index_broadening_preferred_solution**(*env*)

**test_arch_preferred_over_noarch_when_otherwise_equal**(*env*)

**test_noarch_preferred_over_arch_when_version_greater**(*env*)

**test_noarch_preferred_over_arch_when_version_greater_dep**(*env*)

**test_noarch_preferred_over_arch_when_build_greater**(*env*)

**test_noarch_preferred_over_arch_when_build_greater_dep**(*env*)

### Classes

| |
|---|
| *CondaCLIFixture* |
| *PathFactoryFixture* |
| *TmpChannelFixture* |
| *TmpEnvFixture* |

### Functions

| | |
|---|---|
| *conda_cli*(→ collections.abc.Iterator[CondaCLIFixture]) | A function scoped fixture returning CondaCLIFixture instance. |
| *context_aware_monkeypatch*(→ pytest.MonkeyPatch) | A monkeypatch fixture that resets context after each test. |
| *path_factory*(...) | A function scoped fixture returning PathFactoryFixture instance. |
| *tmp_channel*(→ collections.abc.Iterator[TmpChannelFixture]) | A function scoped fixture returning TmpChannelFixture instance. |
| *tmp_env*(→ collections.abc.Iterator[TmpEnvFixture]) | A function scoped fixture returning TmpEnvFixture instance. |
| *tmp_envs_dir*(→ collections.abc.Iterator[pathlib.Path]) | A function scoped fixture returning a temporary environment directory. |
| *tmp_pkgs_dir*(→ collections.abc.Iterator[pathlib.Path]) | A function scoped fixture returning a temporary package cache directory. |
| *conda_move_to_front_of_PATH*() | |

### Attributes

| |
|---|
| *deprecated* |

**deprecated**

**class CondaCLIFixture**

**capsys: pytest.CaptureFixture | None**

**__call__**(**argv: str | os.PathLike[str] | pathlib.Path, raises: type[Exception] | tuple[type[Exception], Ellipsis]*) → tuple[str, str, pytest.ExceptionInfo]

**__call__**(**argv: str | os.PathLike[str] | pathlib.Path*) → tuple[str, str, int]

Test conda CLI. Mimic what is done in *conda.cli.main.main*.

*conda ... == conda_cli(...)*

**Parameters**

- **argv** -- Arguments to parse.
- **raises** -- Expected exception to intercept. If provided, the raised exception will be returned instead of exit code (see pytest.raises and pytest.ExceptionInfo).

**Returns**

Command results (stdout, stderr, exit code or pytest.ExceptionInfo).

**static _cast_args**(*argv: tuple[str | os.PathLike[str] | pathlib.Path, Ellipsis]*) → collections.abc.Iterable[str]

Cast args to string and inspect for *conda run*.

*conda run* is a unique case that requires *--dev* to use the src shell scripts and not the shell scripts provided by the installer.

class PathFactoryFixture

tmp_path: pathlib.Path

__call__(*name: str | None = None*, *prefix: str | None = None*, *suffix: str | None = None*) → pathlib.Path

Unique, non-existent path factory.

Extends pytest's *tmp_path* fixture with a new unique, non-existent path for usage in cases where we need a temporary path that doesn't exist yet.

**Parameters**

- **name** -- Path name to append to *tmp_path*
- **prefix** -- Prefix to prepend to unique name generated
- **suffix** -- Suffix to append to unique name generated

**Returns**

A new unique path

class TmpChannelFixture

path_factory: *PathFactoryFixture*

conda_cli: *CondaCLIFixture*

__call__(**specs: str*) → collections.abc.Iterator[tuple[pathlib.Path, str]]

class TmpEnvFixture

path_factory: *PathFactoryFixture* | pytest.TempPathFactory

conda_cli: *CondaCLIFixture*

get_path() → pathlib.Path

__call__(**packages: str*, *prefix: str | os.PathLike | None = None*) → collections.abc.Iterator[pathlib.Path]

Generate a conda environment with the provided packages.

**Parameters**

- **packages** -- The packages to install into environment
- **prefix** -- The prefix at which to install the conda environment

**Returns**

The conda environment's prefix

conda_cli(*capsys: pytest.CaptureFixture*) → collections.abc.Iterator[*CondaCLIFixture*]

A function scoped fixture returning CondaCLIFixture instance.

Use this for any commands that are local to the current test (e.g., creating a conda environment only used in the test).

context_aware_monkeypatch(*monkeypatch: pytest.MonkeyPatch*) → pytest.MonkeyPatch

A monkeypatch fixture that resets context after each test.

path_factory(*tmp_path: pathlib.Path*) → collections.abc.Iterator[*PathFactoryFixture*]

A function scoped fixture returning PathFactoryFixture instance.

Use this to generate any number of temporary paths for the test that are unique and do not exist yet.

**tmp_channel**(*path_factory:* PathFactoryFixture, *conda_cli:* CondaCLIFixture) → collections.abc.Iterator[*TmpChannelFixture*]

A function scoped fixture returning TmpChannelFixture instance.

**tmp_env**(*path_factory:* PathFactoryFixture, *conda_cli:* CondaCLIFixture) → collections.abc.Iterator[*TmpEnvFixture*]

A function scoped fixture returning TmpEnvFixture instance.

Use this when creating a conda environment that is local to the current test.

**tmp_envs_dir**(*path_factory:* PathFactoryFixture, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary environment directory.

**tmp_pkgs_dir**(*path_factory:* PathFactoryFixture, *mocker: pytest_mock.MockerFixture*) → collections.abc.Iterator[pathlib.Path]

A function scoped fixture returning a temporary package cache directory.

**conda_move_to_front_of_PATH**()

### trust

### constants

Context trust constants.

You could argue that the signatures being here is not necessary; indeed, we are not necessarily going to be able to check them *properly* (based on some prior expectations) as the user, since this is the beginning of trust bootstrapping, the first/backup version of the root of trust metadata. Still, the signatures here are useful for diagnostic purposes, and, more important, to allow self-consistency checks: that helps us avoid breaking the chain of trust if someone accidentally lists the wrong keys down the line. (: The discrepancy can be detected when loading the root data, and we can decline to cache incorrect trust metadata that would make further root updates impossible.

**INITIAL_TRUST_ROOT**

**KEY_MGR_FILE = 'key_mgr.json'**

### signature_verification

Interface between conda-content-trust and conda.

## Classes

*_SignatureVerification*

### Attributes

| |
|---|
| *RE_ROOT_METADATA* |
| *signature_verification* |

**exception SignatureError**

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**RE_ROOT_METADATA**

**class _SignatureVerification**

**property enabled: bool**

**property trusted_root: dict**

**property key_mgr: dict | None**

**_fetch_channel_signing_data**(*signing_data_url: str*, *filename: str*, *etag=None*, *mod_stamp=None*) → dict

**verify**(*repodata_fn: str*, *record:* conda.models.records.PackageRecord)

**__call__**(*repodata_fn: str*, *unlink_precs: tuple[*conda.models.records.PackageRecord*, Ellipsis]*, *link_precs: tuple[*conda.models.records.PackageRecord*, Ellipsis]*) → None

**classmethod cache_clear**() → None

**signature_verification**

**deprecated**

## utils

Utility functions.

### Functions

| | |
|---|---|
| unix_path_to_win(path[, root_prefix]) | Convert a path or :-separated string of paths into a Windows representation |
| win_path_to_cygwin(path) | |
| cygwin_path_to_win(path) | |
| translate_stream(stream, translator) | |
| human_bytes(n) | Return the number of bytes n in more human readable form. |
| sys_prefix_unfollowed() | Since conda is installed into non-root environments as a symlink only |
| quote_for_shell(*arguments) | Properly quote arguments for command line passing. |
| massage_arguments(arguments[, errors]) | |
| wrap_subprocess_call(root_prefix, prefix, dev_mode, ...) | |
| get_comspec() | Returns COMSPEC from envvars. |
| ensure_dir_exists(func) | Ensures that the directory exists for functions returning |

### Attributes

| |
|---|
| _UNIX_SHELL_BASE |
| _MSYS2_SHELL_BASE |
| _SHELLS |
| _RE_UNSAFE |

**unix_path_to_win**(*path*, *root_prefix=''*)

Convert a path or :-separated string of paths into a Windows representation

Does not add cygdrive. If you need that, set root_prefix to "/cygdrive"

**win_path_to_cygwin**(*path*)

**cygwin_path_to_win**(*path*)

**translate_stream**(*stream*, *translator*)

**human_bytes**(*n*)

Return the number of bytes n in more human readable form.

### Examples

```
>>> human_bytes(42)
'42 B'
>>> human_bytes(1042)
'1 KB'
>>> human_bytes(10004242)
'9.5 MB'
>>> human_bytes(100000004242)
'93.13 GB'
```

**_UNIX_SHELL_BASE**

**_MSYS2_SHELL_BASE**

**_SHELLS**

**sys_prefix_unfollowed**()

Since conda is installed into non-root environments as a symlink only and because sys.prefix follows symlinks, this function can be used to get the 'unfollowed' sys.prefix.

This value is usually the same as the prefix of the environment into which conda has been symlinked. An example of when this is necessary is when conda looks for external sub-commands in find_commands.py

**quote_for_shell**(**arguments*)

Properly quote arguments for command line passing.

For POSIX uses *shlex.join*, for Windows uses a custom implementation to properly escape metacharacters.

**Parameters**
**arguments** (*list of str*) -- Arguments to quote.

**Returns**
Quoted arguments.

**Return type**
str

**_RE_UNSAFE**

**massage_arguments**(*arguments*, *errors='assert'*)

**wrap_subprocess_call**(*root_prefix*, *prefix*, *dev_mode*, *debug_wrapper_scripts*, *arguments*, *use_system_tmp_path=False*)

**get_comspec**()

Returns COMSPEC from envvars.

Ensures COMSPEC envvar is set to cmd.exe, if not attempt to find it.

**Raises**
**KeyError** -- COMSPEC is undefined and cannot be found.

**Returns**
COMSPEC value.

**Return type**
str

**ensure_dir_exists**(*func*)

Ensures that the directory exists for functions returning a Path object containing a directory

### Functions

| |
|---|
| *conda_signal_handler*(signum, frame) |

### Attributes

| |
|---|
| *__version__* |
| *__name__* |
| *__author__* |
| *__email__* |
| *__license__* |
| *__copyright__* |
| *__summary__* |
| *__url__* |
| *CONDA_PACKAGE_ROOT* |

**__version__**

**__name__ = 'conda'**

**__author__ = 'Anaconda, Inc.'**

**__email__ = 'conda@continuum.io'**

**__license__ = 'BSD-3-Clause'**

**__copyright__ = 'Copyright (c) 2012, Anaconda, Inc.'**

**__summary__**

**__url__ = 'https://github.com/conda/conda'**

**CONDA_PACKAGE_ROOT**

**exception CondaError**(*message: str | None*, *caused_by: Any = None*, ***kwargs*)

Bases: Exception

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**return_code: int = 1**

**reportable: bool = False**

**__repr__**() → str

Return repr(self).

**__str__**() → str

Return str(self).

**dump_map**() → dict[str, Any]

**exception CondaMultiError**(*errors: collections.abc.Iterable[*CondaError*]*)

Bases: *CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**__repr__**() → str

Return repr(self).

**__str__**() → str

Return str(self).

**dump_map**() → dict[str, str | tuple[str, Ellipsis]]

**contains**(*exception_class: BaseException | tuple[BaseException]*) → bool

**exception CondaExitZero**(*message: str | None*, *caused_by: Any = None*, ***kwargs*)

Bases: *CondaError*

Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

**return_code = 0**

**conda_signal_handler**(*signum: int*, *frame: Any*)

### conda_env

### cli

### installers

# PYTHON MODULE INDEX

## C

# INDEX

## Symbols

## A

## B

## C

## D

## E

## F

## G

## H



## I

## J



## K



## L

## M

## N

## O

## P

## Q

## R

## S

## T

## U

## V

## W

## X



## Y



## Z