conda Documentation

Release 23.3.1.post2+bdcba5dd0

Anaconda, Inc

Jul 17, 2023

CONTENTS

Package, dependency and environment management for any language---Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN

Conda is an open-source package management system and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. Conda easily creates, saves, loads, and switches between environments on your local computer. It was created for Python programs but it can package and distribute software for any language.

Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

In its default configuration, conda can install and manage the over 7,500 packages at repo.anaconda.com that are built, reviewed, and maintained by Anaconda®.

Conda can be combined with continuous integration systems such as Travis CI and AppVeyor to provide frequent, automated testing of your code.

The conda package and environment manager is included in all versions of *Anaconda*®, *Miniconda*, and Anaconda Repository. Conda is also included in Anaconda Enterprise, which provides on-site enterprise package and environment management for Python, R, Node.js, Java, and other application stacks. Conda is also available on conda-forge, a community channel. You may also get conda on PyPI, but that approach may not be as up to date.

CHAPTER

ONE

USER GUIDE

1.1 Concepts

Explore the links to learn more about conda foundations.

1.1.1 Conda 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 *Command reference*. You can see the same information at the command line by *viewing the command-line help*.

1.1.2 Conda packages

- *More information*

What is a conda package?

A conda 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 installer. The command `conda create -n envname anaconda` creates an environment that exactly matches what would be created from the Anaconda 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

Go deeper on how to *manage packages*. Learn more about package metadata, repository structure and index, and package match specifications at *Package specifications*.

1.1.3 Conda 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",
      "path_type": "hardlink",
      "sha256": "76f3b6e34feeb651aff33ca59e0279c4eadce5a50c6ad93b961c846f7ba717e9",
      "size_in_bytes": 2067
```

(continues on next page)

(continued from previous page)

```
    },
    {
      "_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: • A build with other dependencies, such as Python 3.4 instead of Python 2.7. • A bug fix in the build process. • 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. EXAMPLE: `x86_64` Conda currently does not use this key.
platform	string	Optional. The OS that the package is built for. EXAMPLE: `osx` 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
```

(continues on next page)

(continued from previous page)

```
< 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
```

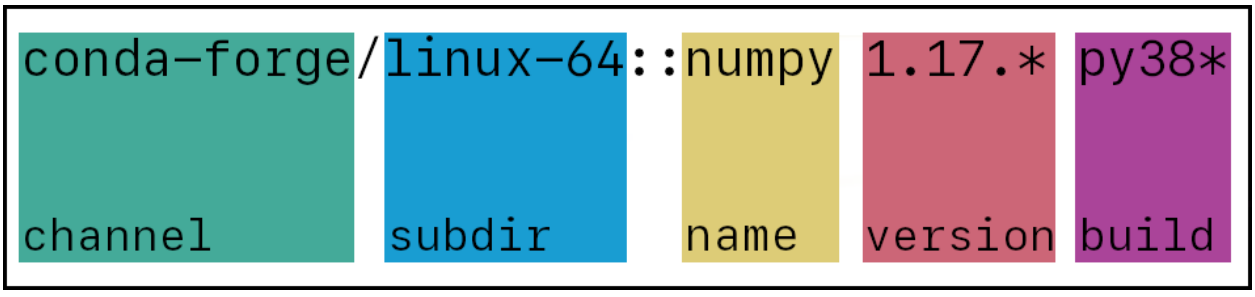
1.1.4 Package search and install specifications

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 preceeding 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*]"
```

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"
```

1.1.5 Conda channels

What is a "conda channel"?

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 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>
```

1.1.6 Conda environments

A conda environment is a directory that contains a specific collection of conda 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 that Anaconda or Miniconda 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?

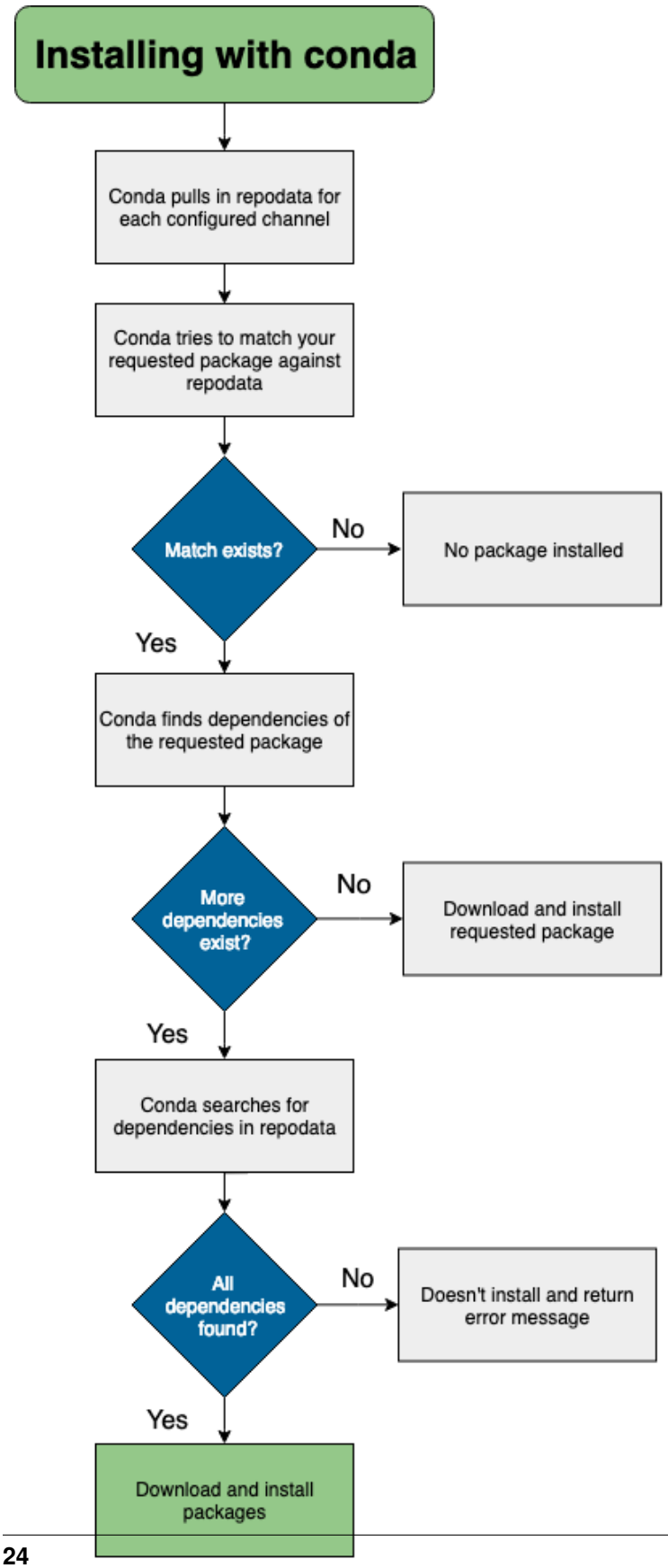
Package system differentiators

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.

1.1.7 Installing with conda



To install conda packages in the terminal or an Anaconda Prompt, run:

```
conda install [packagename]
```

During the install process, files are extracted into the specified environment, defaulting to the current environment if none is specified. Installing the files of a conda package into an environment can be thought of as changing the directory to an environment, and then downloading and extracting the artifact and its dependencies---all with the single `conda install [packagename]` command.

Read more about *conda environments and directory structure*.

- When you `conda install` a package that exists in a channel and has no dependencies, conda:
  - Looks at your configured channels (in priority).
  - Reaches out to the repodata associated with your channels/platform.
  - Parses repodata to search for the package.
  - Once the package is found, conda pulls it down and installs.

Conda update versus conda install

`conda update` is used to update 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
```

1.1.8 Conda 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:

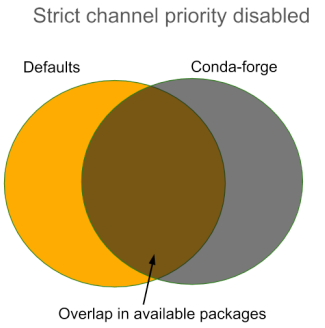
- **Using conda-forge?**
  - Use conda-metachannel to reduce conda's problem size.
- **Using bioconda?**
  - Use conda-metachannel to reduce conda's problem size.
  - Read more about docker images.
- **Specifying very broad package specs?**

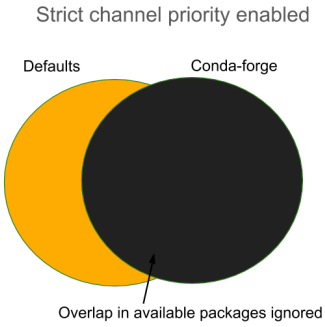
  - Be more specific. Letting conda filter more candidates makes it faster. For example, instead of `numpy`, we recommend `numpy=1.15` or, even better, `numpy=1.15.4`.
  - If you are using R, instead of specifying only `r-essentials`, specify `r-base=3.5 r-essentials`.
- **Feeling frustrated with "verifying 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.

1.1.9 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.

1.1.10 Conda 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

```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!

When you're comfortable with the conda concepts, learn how to *get started using conda*.

1.2 Getting started with conda

Conda is a powerful package manager and environment manager that you use with command line commands at the Anaconda Prompt for Windows, or in a terminal window for macOS or Linux.

This 20-minute guide to getting started with conda lets you try out the major features of conda. You should understand how conda works when you finish this guide.

SEE ALSO: Getting started with Anaconda Navigator, a graphical user interface that lets you use conda in a web-like interface without having to enter manual commands. Compare the Getting started guides for each to see which program you prefer.

1.2.1 Before you start

You should have already installed Anaconda.

1.2.2 Contents

- *Starting conda* on Windows, macOS, or Linux. 2 MINUTES
- *Managing conda*. Verify that Anaconda is installed and check that conda is updated to the current version. 3 MINUTES
- *Managing environments*. Create *environments* and move easily between them. 5 MINUTES
- *Managing Python*. Create an environment that has a different version of Python. 5 MINUTES
- *Managing packages*. Find packages available for you to install. Install packages. 5 MINUTES

TOTAL TIME: 20 MINUTES

1.2.3 Starting conda

Windows

- From the Start menu, search for and open "Anaconda Prompt."

On Windows, all commands below are typed into the Anaconda Prompt window.

MacOS

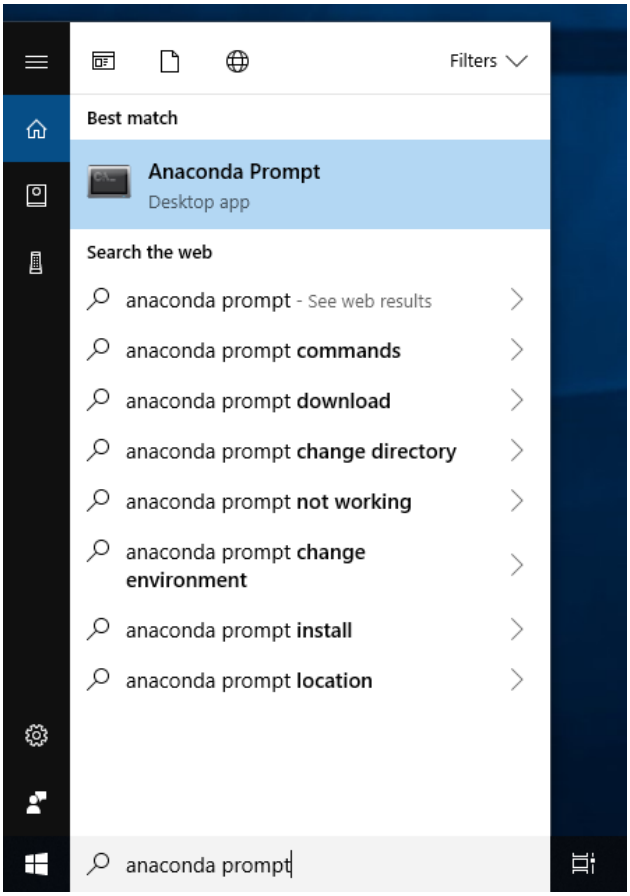
- Open Launchpad, then click the terminal icon.

On macOS, all commands below are typed into the terminal window.

Linux

- Open a terminal window.

On Linux, all commands below are typed into the terminal window.


Filters
Best match
Anaconda Prompt
Desktop app
Search the web
anaconda prompt - See web results
anaconda prompt commands
anaconda prompt download
anaconda prompt change directory
anaconda prompt not working
anaconda prompt change environment
anaconda prompt install
anaconda prompt location
anaconda prompt

1.2.4 Managing conda

Verify that conda is installed and running on your system by typing:

```
conda --version
```

Conda displays the number of the version that you have installed. You do not need to navigate to the Anaconda directory.

EXAMPLE: `conda 4.7.12`

Note: If you get an error message, make sure you closed and re-opened the terminal window after installing, or do it now. Then verify that you are logged into the same user account that you used to install Anaconda or Miniconda.

Update conda to the current version. Type the following:

```
conda update conda
```

Conda compares versions and then displays what is available to install.

If a newer version of conda is available, type `y` to update:

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

Tip: We recommend that you always keep conda updated to the latest version.

1.2.5 Managing environments

Conda allows you to create separate environments containing files, packages, and their dependencies that will not interact with other environments.

When you begin using conda, you already have a default environment named `base`. You don't want to put programs into your base environment, though. Create separate environments to keep your programs isolated from each other.

1. Create a new environment and install a package in it.

   We will name the environment `snowflakes` and install the package BioPython. At the Anaconda Prompt or in your terminal window, type the following:

   ```
   conda create --name snowflakes biopython
   ```

   Conda checks to see what additional packages ("dependencies") BioPython will need, and asks if you want to proceed:

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

   Type "y" and press Enter to proceed.

2. To use, or "activate" the new environment, type the following:

   - Windows: `conda activate snowflakes`
   - macOS and Linux: `conda activate snowflakes`

   Note: `conda activate` only works on conda 4.6 and later versions.

For conda versions prior to 4.6, type:

- Windows: `activate snowflakes`
- macOS and Linux: `source activate snowflakes`

Now that you are in your `snowflakes` environment, any conda commands you type will go to that environment until you deactivate it.

3. To see a list of all your environments, type:

```
conda info --envs
```

A list of environments appears, similar to the following:

```
conda environments:

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

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

4. Change your current environment back to the default (base): `conda activate`

Note: For versions prior to conda 4.6, use:

- Windows: `activate`
- macOS, Linux: `source activate`

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

1.2.6 Managing Python

When you create a new environment, conda installs the same Python version you used when you downloaded and installed Anaconda. If you want to use a different version of Python, for example Python 3.5, simply create a new environment and specify the version of Python that you want.

1. Create a new environment named "snakes" that contains Python 3.9:

```
conda create --name snakes python=3.9
```

When conda asks if you want to proceed, type "y" and press Enter.

2. Activate the new environment:

- Windows: `conda activate snakes`
- macOS and Linux: `conda activate snakes`

Note: `conda activate` only works on conda 4.6 and later versions.

For conda versions prior to 4.6, type:

- Windows: `activate snakes`
- macOS and Linux: `source activate snakes`

3. Verify that the snakes environment has been added and is active:

```
conda info --envs
```

Conda displays the list of all environments with an asterisk (*) after the name of the active environment:

```
# conda environments:
#
base                     /home/username/anaconda3
snakes                *  /home/username/anaconda3/envs/snakes
snowflakes               /home/username/anaconda3/envs/snowflakes
```

The active environment is also displayed in front of your prompt in (parentheses) or [brackets] like this:

```
(snakes) $
```

4. Verify which version of Python is in your current environment:

```
python --version
```

5. Deactivate the snakes environment and return to base environment: `conda activate`

Note: For versions prior to conda 4.6, use:

- Windows: `activate`
- macOS, Linux: `source activate`

1.2.7 Managing packages

In this section, you check which packages you have installed, check which are available and look for a specific package and install it.

1. To find a package you have already installed, first activate the environment you want to search. Look above for the commands to *activate your snakes environment*.
2. Check to see if a package you have not installed named "beautifulsoup4" is available from the Anaconda repository (must be connected to the Internet):

```
conda search beautifulsoup4
```

Conda displays a list of all packages with that name on the Anaconda repository, so we know it is available.

3. Install this package into the current environment:

```
conda install beautifulsoup4
```

4. Check to see if the newly installed program is in this environment:

```
conda list
```

1.2.8 More information

- *Conda cheat sheet*
- Full documentation--- https://conda.io/docs/
- Free community support--- https://groups.google.com/a/anaconda.com/forum/#!forum/anaconda
- Paid support options--- https://www.anaconda.com/support/

1.3 Installation

The fastest way to *obtain* conda is to install *Miniconda*, a mini version of *Anaconda* that includes only conda and its dependencies. If you prefer to have conda plus over 7,500 open-source packages, install Anaconda.

We recommend you install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. You can also install Anaconda system wide, which does require administrator permissions.

For information on using our graphical installers for Windows or macOS, see the instructions for installing Anaconda.

1.3.1 System requirements

- 32- or 64-bit computer
- For Miniconda: 400 MB disk space
- For Anaconda: Minimum 3 GB disk space to download and install
- Windows, macOS, or Linux
- For Windows: Windows 8.1 or newer for Python 3.9, or Windows Vista or newer for Python 3.8

Note: You do not need administrative or root permissions to install Anaconda if you select a user-writable install location.

1.3.2 Regular installation

Follow the instructions for your operating system:

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

1.3.3 Installing in silent mode

You can use *silent installation* of Miniconda or Anaconda for deployment or testing or building services such as Travis CI and AppVeyor.

Follow the silent-mode instructions for your operating system:

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

1.3.4 Installing conda on a system that has other Python installations or packages

You do not need to uninstall other Python installations or packages in order to use conda. Even if you already have a system Python, another Python installation from a source such as the macOS Homebrew package manager and globally installed packages from pip such as pandas and NumPy, you do not need to uninstall, remove, or change any of them before using conda.

Install Anaconda or Miniconda normally, and let the installer add the conda installation of Python to your PATH environment variable. There is no need to set the PYTHONPATH environment variable.

To see if the conda installation of Python is in your PATH variable:

- On Windows, open an Anaconda Prompt and run `echo %PATH%`
- On macOS and Linux, open the terminal and run `echo $PATH`

To see which Python installation is currently set as the default:

- On Windows, open an Anaconda Prompt and run `where python`
- On macOS and Linux, open the terminal and run `which python`

To see which packages are installed in your current conda environment and their version numbers, in your terminal window or an Anaconda Prompt, run `conda list`.

Downloading conda

You have 3 conda download options:

- Download Anaconda---free.
- Download Miniconda---free.
- Purchase Anaconda Enterprise.

You can download any of these 3 options with legacy Python 2.7 or current Python 3.

You can also choose a version with a GUI or a command line installer.

Tip: If you are unsure which option to download, choose the most recent version of Anaconda3. If you are on Windows or macOS, choose the version with the GUI installer.

Anaconda or Miniconda?

Choose Anaconda if you:

- Are new to conda or Python.
- Like the convenience of having Python and over 1,500 scientific packages automatically installed at once.
- Have the time and disk space---a few minutes and 3 GB.
- Do not want to individually install each of the packages you want to use.
- Wish to use a set of packages curated and vetted for interoperability and usability.

Choose Miniconda if you:

- Do not mind installing each of the packages you want to use individually.
- Do not have time or disk space to install over 1,500 packages at once.
- Want fast access to Python and the conda commands and you wish to sort out the other programs later.

Choosing a version of Anaconda or Miniconda

- Whether you use Anaconda or Miniconda, select the most recent version.
- Select an older version from the archive only if you are testing or need an older version for a specific purpose.
- To use conda on Windows XP, select Anaconda 2.3.0 and see ../configuration/use-winxp-with-proxy.

GUI versus command line installer

Both GUI and command line installers are available for Windows, macOS, and Linux:

- If you do not wish to enter commands in a terminal window, choose the GUI installer.
- If GUIs slow you down, choose the command line version.

Choosing a version of Python

- The last version of Python 2 is 2.7, which is included with Anaconda and Miniconda.
- The newest stable version of Python is quickly included with Anaconda3 and Miniconda3.
- You can easily set up additional versions of Python such as 3.9 by downloading any version and creating a new environment with just a few clicks. See *Getting started with conda*.

Cryptographic hash verification

SHA-256 checksums are available for Miniconda and Anaconda. 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 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 the Anaconda Prompt.
5. Test your installation. In your terminal window or Anaconda Prompt, 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. For Anaconda, substitute `Anaconda` for `Miniconda` in all of the commands.

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 your Anaconda Prompt from the start menu.
2. Navigate to the `anaconda` directory.
3. 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

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

     ```
     bash Miniconda3-latest-MacOSX-x86_64.sh
     ```

   - Anaconda---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. Test your installation. In your terminal window or Anaconda Prompt, 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. For Anaconda, substitute `Anaconda` for `Miniconda` in all of the commands.

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
```

The installer prompts "Do you wish the installer to initialize Miniconda3 by running `conda init`?" We recommend "yes".

Note: If you enter "no", then conda will not modify your shell scripts at all. In order to initialize after the installation process is done, first run `source <path to conda>/bin/activate` and then run `conda init`.

macOS Catalina (and later)

If you are on macOS Catalina (or later versions), the default shell is zsh. You will instead need to run `source <path to conda>/bin/activate` followed by `conda init zsh` (to explicitly select the type of shell to initialize).

Updating Anaconda or Miniconda

1. Open a terminal window.
2. Navigate to the `anaconda` directory.
3. Run `conda update conda`.

Uninstalling Anaconda or Miniconda

1. Open a terminal window.
2. Remove the entire Miniconda install directory with:

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

3. OPTIONAL: Edit `~/.bash_profile` to remove the Miniconda directory from your PATH environment variable.
4. 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 installer for Linux.
2. *Verify your installer hashes*.
3. In your terminal window, run:
   - Miniconda:

     ```
     bash Miniconda3-latest-Linux-x86_64.sh
     ```

   - Anaconda:

     ```
     bash Anaconda-latest-Linux-x86_64.sh
     ```

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 or Anaconda Prompt, 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:

```
conda init fish
```

Installing in silent mode

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

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:

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

3. OPTIONAL: Edit `~/.bash_profile` to remove the Miniconda directory from your PATH environment variable.
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, the package manager from Anaconda, 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]
```

(continues on next page)

(continued from previous page)

```
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

# 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
```

(continues on next page)

(continued from previous page)

```
# 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
```

(continues on next page)

(continued from previous page)

```
$HOME/.conda/pkgs
envs_dirs:
/shared/conda/envs
$HOME/.conda/envs
```

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

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

If you're new to conda, check out the documentation at https://conda.io/docs/

1.4 Configuration

1.4.1 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 Anaconda Prompt or 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 or Anaconda Prompt. 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 or Anaconda Prompt by running `conda config --help`. You can also see the conda channel configuration for more information.

Tip: Conda supports *tab completion* with external packages instead of internal configuration.

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",
```

(continues on next page)

(continued from previous page)

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

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.

General configuration

- *Channel locations (channels)*
- *Default channels (default_channels)*
- *Update conda automatically (auto_update_conda)*
- *Always yes (always_yes)*
- *Show channel URLs (show_channel_urls)*
- *Change command prompt (changeps1)*
- *Add pip as Python dependency (add_pip_as_python_dependency)*
- *Use pip (use_pip)*
- *Configure conda for use behind a proxy server (proxy_servers)*
- *SSL verification (ssl_verify)*
- *Offline mode only (offline)*

Channel locations (channels)

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

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

Update conda automatically (auto_update_conda)

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

EXAMPLE:

```
show_channel_urls: True
```

Change command prompt (changeps1)

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
```

Configure conda for use behind a proxy server (proxy_servers)

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

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 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 verification (ssl_verify)

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
```

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 mode only (offline)

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

EXAMPLE:

```
offline: True
```

Advanced configuration

- *Disallow soft-linking (allow_softlinks)*
- *Set a channel alias (channel_alias)*
- *Always add packages by default (create_default_packages)*
- *Track features (track_features)*
- *Disable updating of dependencies (update_dependencies)*
- *Disallow installation of specific packages (disallow)*
- *Add Anaconda.org token to automatically see private packages (add_anaconda_token)*
- *Specify environment directories (envs_dirs)*
- *Specify package directories (pkgs_dirs)*
- *Force conda to download only .tar.bz2 packages (use_only_tar_bz2)*

Disallow soft-linking (allow_softlinks)

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 `True`.

EXAMPLE:

```
allow_softlinks: False
```

Set a channel alias (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>
```

Always add packages by default (create_default_packages)

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
```

Disable updating of dependencies (update_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 installation of specific packages (disallow)

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

EXAMPLE:

```
disallow:
  - anaconda
```

Add Anaconda.org token to automatically see private packages (add_anaconda_token)

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.

Specify environment directories (envs_dirs)

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`

Specify package directories (pkgs_dirs)

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`

Force conda to download only .tar.bz2 packages (use_only_tar_bz2)

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.

Conda-build configuration

- *Specify conda-build output root directory (root-dir)*
- *Specify conda-build build folder (conda-build 3.16.3+) (output_folder)*
- pkg_format
- *Automatically upload conda-build packages to Anaconda.org (anaconda_upload)*
- *Token to be used for Anaconda.org uploads (conda-build 3.0+) (anaconda_token)*
- *Limit build output verbosity (conda-build 3.0+) (quiet)*
- *Disable filename hashing (conda-build 3.0+) (filename_hashing)*
- *Disable recipe and package verification (conda-build 3.0+) (no_verify)*
- *Disable per-build folder creation (conda-build 3.0+) (set_build_id)*
- *Skip building packages that already exist (conda-build 3.0+) (skip_existing)*
- *Omit recipe from package (conda-build 3.0+) (include_recipe)*
- *Disable activation of environments during build/test (conda-build 3.0+) (activate)*
- *Disable long prefix during test (conda-build 3.16.3+) (long_test_prefix)*
- *PyPI upload settings (conda-build 3.0+) (pypirc)*
- *PyPI repository to upload to (conda-build 3.0+) (pypi_repository)*
- *Configuring number of threads*

Specify conda-build output root directory (root-dir)

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
```

Specify conda-build build folder (conda-build 3.16.3+) (output_folder)

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 the root build directory (`root-dir`).

```
conda-build:
    output_folder: conda-bld
```

Specify conda-build package version (pkg_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
```

Automatically upload conda-build packages to Anaconda.org (anaconda_upload)

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

EXAMPLE:

```
anaconda_upload: True
```

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

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
```

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

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

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

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

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.

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

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
```

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

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 building packages that already exist (conda-build 3.0+) (skip_existing)

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
```

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

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.

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

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
```

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

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`.

PyPI upload settings (conda-build 3.0+) (pypirc)

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 to upload to (conda-build 3.0+) (pypi_repository)

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`

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
```

Obtaining information from the .condarc file

Note: It may be necessary to add the "force" option `-f` to the following commands.

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 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*.

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
```

1.4.2 Sample .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
```

1.4.3 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.

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*.

1.4.4 Administering a multi-user conda installation

By default, conda and all packages it installs, including Anaconda, 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:

   ```
   $ 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.

1.4.5 Enabling tab completion

Conda versions up to 4.3 supports tab completion in Bash shells via the argcomplete package. Bash tab completion has been removed starting with version 4.4.0.

To enable tab completion in your Bash shell:

1. Make sure that argcomplete is installed:

```
conda install argcomplete
```

2. Add the following code to your bash profile:

```
eval "$(register-python-argcomplete conda)"
```

3. Test it:

1. Open a new terminal window or an Anaconda Prompt.
2. Type: `conda ins`, and then press the Tab key.

   The command completes to:

```
conda install
```

To get tab completion in Zsh, see conda-zsh-completion.

1.4.6 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 pip_interop_enabled True
```

Note: Setting `pip_interop_enabled` 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
```

1.4.7 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.

1.4.8 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.

1.5 Tasks

1.5.1 Managing conda

Verifying that conda is installed

To verify that conda is installed, in your terminal window or an Anaconda Prompt, 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 or an Anaconda Prompt:

```
conda info
```

OR

```
conda -V
```

Updating conda to the current version

To update conda, in your terminal window or an Anaconda Prompt, 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 or Anaconda Prompt: `conda config --set notify_outdated_conda false`

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

1.5.2 Managing environments

- *Removing an environment*

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.

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

Note: `conda activate` and `conda deactivate` only work on conda 4.6 and later versions. For conda versions prior to 4.6, run:

- Windows: `activate` or `deactivate`
- Linux and macOS: `source activate` or `source deactivate`

Creating an environment with commands

Tip: By default, environments are installed into the `envs` directory in your conda directory. See *Specifying a location for an environment* or run `conda create --help` for information on specifying a different path.

Use the terminal or an Anaconda Prompt for the following steps:

1. To create an environment:

   ```
   conda create --name myenv
   ```

   Note: Replace `myenv` with the environment name.

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 1 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 or an Anaconda Prompt 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 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 --prefix ./env --file environment.yml  --prune
```

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

Cloning an environment

Use the terminal or an Anaconda Prompt 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 or an Anaconda Prompt 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
```

(continues on next page)

(continued from previous page)

```
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
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*.

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

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 our entries. In the latter case, you do not. We do not recommend multi-user installs.

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

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 base` in Anaconda Prompt.

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. Removing the need to modify PATH makes conda less disruptive to other software on your system. For more information, read the output from `conda init --help`.

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

```
auto_activate_base: bool
```

This setting controls whether or not conda activates your base 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 environment will be available until the environment is activated with `conda activate base`. 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 base 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 base 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 base 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 or an Anaconda Prompt 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 or an Anaconda Prompt, 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 or an Anaconda Prompt, run:

  ```
  conda list -n myenv
  ```

- If the environment is activated, in your terminal window or an Anaconda Prompt, run:

  ```
  conda list
  ```

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

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

Using pip in an environment

To use pip in your environment, in your terminal window or an Anaconda Prompt, 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 Anaconda Prompt 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
  - 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 or an Anaconda Prompt, 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 or an Anaconda Prompt, run:

```
conda info --envs
```

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

1.5.3 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: false` to your `.condarc` file.

  OR

- Run the equivalent command:

```
conda config --set channel_priority false
```

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'.
```

(continues on next page)

(continued from previous page)

```
channel_priority: flexible
```

1.5.4 Creating custom channels

Channels are the path that conda takes to look for packages. The easiest way to use and manage custom channels is to use a private or public repository on Anaconda.org. If you designate your Anaconda.org repository as private, then only you and those you grant access to can access your private repository.

If you do not wish to upload your packages to the Internet, you can build a custom repository served either through a web server or locally using a `file://` URL.

To create a custom channel:

1. If you have not yet used conda-build, install conda-build:

   ```
   conda install conda-build
   ```

2. Organize all the packages in subdirectories for the platforms you wish to serve:

   ```
    channel/
   linux-64/
    package-1.0-0.tar.bz2
   linux-32/
    package-1.0-0.tar.bz2
   osx-64/
    package-1.0-0.tar.bz2
   win-64/
    package-1.0-0.tar.bz2
   win-32/
    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.

Note: 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.

1.5.5 Managing packages

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

Searching for packages

Use the terminal or an Anaconda Prompt 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 or an Anaconda Prompt 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 or an Anaconda Prompt, 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 or an Anaconda Prompt, 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:
   - On Windows, in your Anaconda Prompt, run `activate myenv`.
   - On macOS and Linux, in your terminal window, run `conda activate myenv`.
2. To use pip to install a program such as See, in your terminal window or an Anaconda Prompt, run:

   ```
   pip install see
   ```

3. To verify the package was installed, in your terminal window or an Anaconda Prompt, 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 or an Anaconda Prompt, 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 or an Anaconda Prompt 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 or an Anaconda Prompt 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 3.8 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 or an Anaconda Prompt, 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 Anaconda Prompt or terminal 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 or an Anaconda Prompt 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
  ```

1.5.6 Managing Python

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

Anaconda supports Python 3.7, 3.8, 3.9 and 3.10. The current default is Python 3.9.

Viewing a list of available Python versions

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

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

     Note: Replace `py39` with the name of the environment you want to create. `anaconda` is the metapackage that includes all of the Python packages comprising the Anaconda distribution. `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 or an Anaconda Prompt, 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 or upgrading Python

Use the terminal or an Anaconda Prompt for the following steps.

If you are in an environment with Python version 3.4.2, the following command updates Python to the latest version in the 3.4 branch:

```
conda update python
```

The following command upgrades Python to another branch---3.8---by installing that version of Python. It is not recommended, rather it is preferable to create a new environment. The resolver has to work very hard to determine exactly which packages to upgrade. But it is possible, and the command is:

```
conda install python=3.8
```

1.5.7 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.

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 or an Anaconda Prompt 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
       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.

1.5.8 Using conda with Travis CI

If you are already using Travis CI, using conda is a preferable alternative to using apt and pip to install packages. The Debian repos provided by Travis may not include packages for all versions of Python or may not be updated as quickly. Installing such packages with pip may also be undesirable, as this can take a long time, which can consume a large portion of the 50 minutes that Travis allows for each build. Using conda also lets you test the building of conda recipes on Travis.

This page describes how to use conda to test a Python package on Travis CI. However, you can use conda with any language, not just Python.

The .travis.yml file

The following code sample shows how to modify the `.travis.yml` file to use Miniconda for a project that supports Python 3.7, 3.8, 3.9 and 3.10:

```
language: python
python:
  # We don't actually use the Travis Python, but this keeps it organized.
  - "3.7"
  - "3.8"
  - "3.9"
  - "3.10"
install:
  # We do this conditionally because it saves us some downloading if the
  # version is the same.
  - wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O␣
↪miniconda.sh
  - bash miniconda.sh -b -p $HOME/miniconda
  - source "$HOME/miniconda/etc/profile.d/conda.sh"
  - hash -r
  - conda config --set always_yes yes --set changeps1 no
  - conda update -q conda
  # Useful for debugging any issues with conda
  - conda info -a

  # Replace dep1 dep2 ... with your dependencies
  - conda create -q -n test-environment python=$TRAVIS_PYTHON_VERSION dep1 dep2 ...
  - conda activate test-environment
  - python -m pip install .

script:
  # Your test script goes here
```

Note: For information about the basic configuration for Travis CI, see Building a Python Project.

Supporting packages that do not have official builds

To support a package that does not have official Anaconda builds:

1. Build the package yourself.
2. Add it to an Anaconda.org channel.
3. Add the following line to the install steps in `.travis.yml` so that it finds the packages on that channel:

```
- conda config --add channels your_Anaconda_dot_org_username
```

Note: Replace `your_Anaconda_dot_org_username` with your user name.

Building a conda recipe

If you support official conda packages for your project, you may want to use conda-build in Travis, so the building of your recipe is tested as well.

1. Include the conda recipe in the same directory as your source code.
2. In your `.travis.yml` file, replace the following line:

```
- python setup.py install
```

with these lines:

```
- conda build your-conda-recipe
- conda install your-package --use-local
```

1.5.9 Viewing command-line help

To see a list of supported conda commands, in your terminal window or an Anaconda Prompt, 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 or an Anaconda Prompt, run:

```
conda create -h
```

Note: You can see the same command help in *Command reference*.

1.6 Cheat sheet

See the `conda cheatsheet` PDF (1 MB) for a single-page summary of the most important information about using conda (link always points to the latest version).

1.6.1 Versions

- `conda 4.14.x` (latest)
- `conda 4.12.x`
- `conda 4.6.x`

1.7 Troubleshooting

1.7.1 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

1.7.2 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 (Anaconda Prompt, for example) 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 an Anaconda Prompt 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.

1.7.3 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/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 "Anaconda Prompt" or 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 `--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
   # #   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, or (2) a path
   # #   to a directory containing certificates of trusted CA.
   # #
   # 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, or (2) a path
# #   to a directory containing certificates of trusted CA.
# #
# 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`

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.

1.7.4 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.

1.7.5 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.

1.7.6 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.

1.7.7 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`.

1.7.8 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.

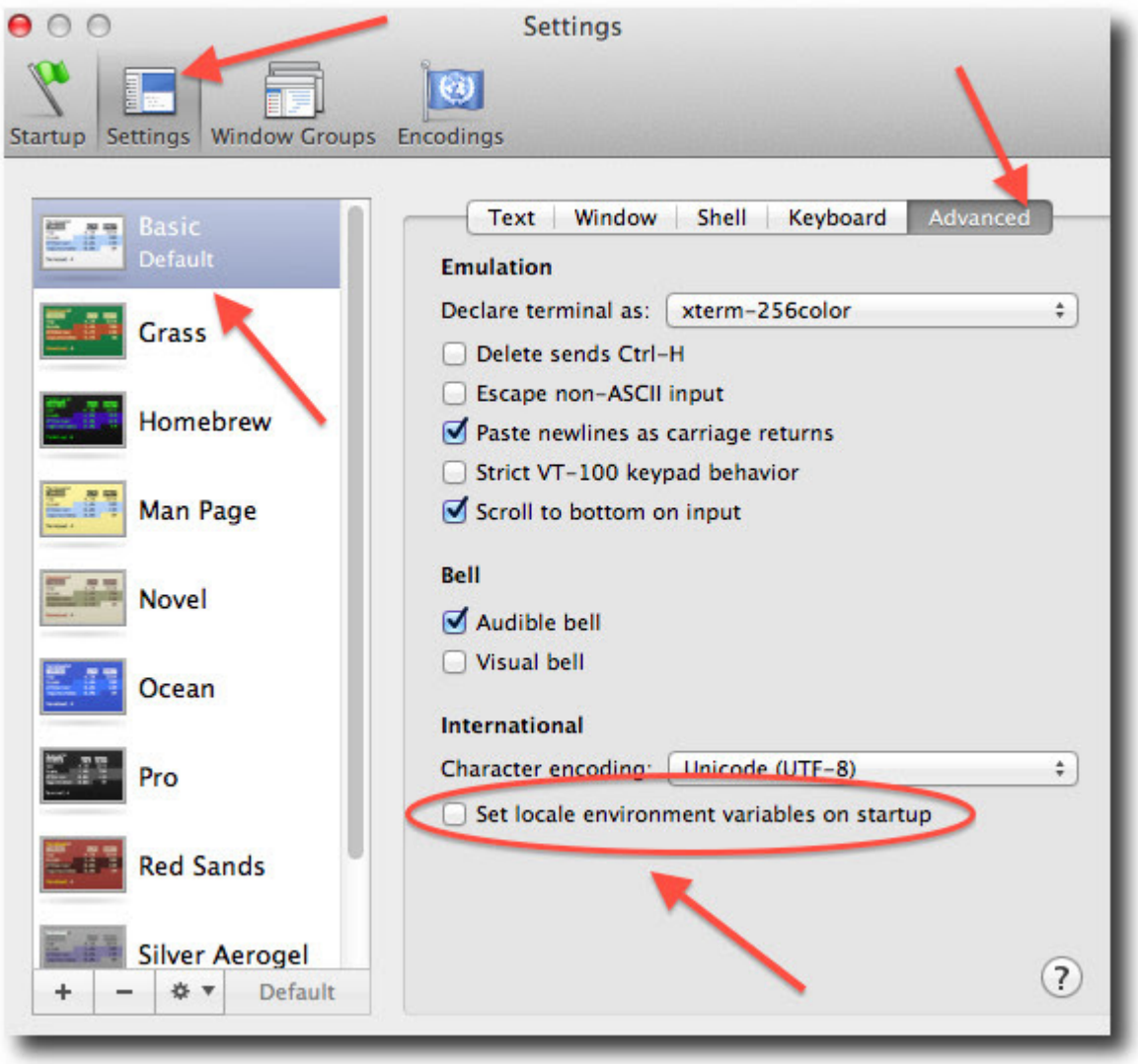
1.7.9 macOS error "ValueError unknown locale: UTF-8"

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.

1.7.10 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.

1.7.11 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 root 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.

1.7.12 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.

1.7.13 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
```

1.7.14 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:

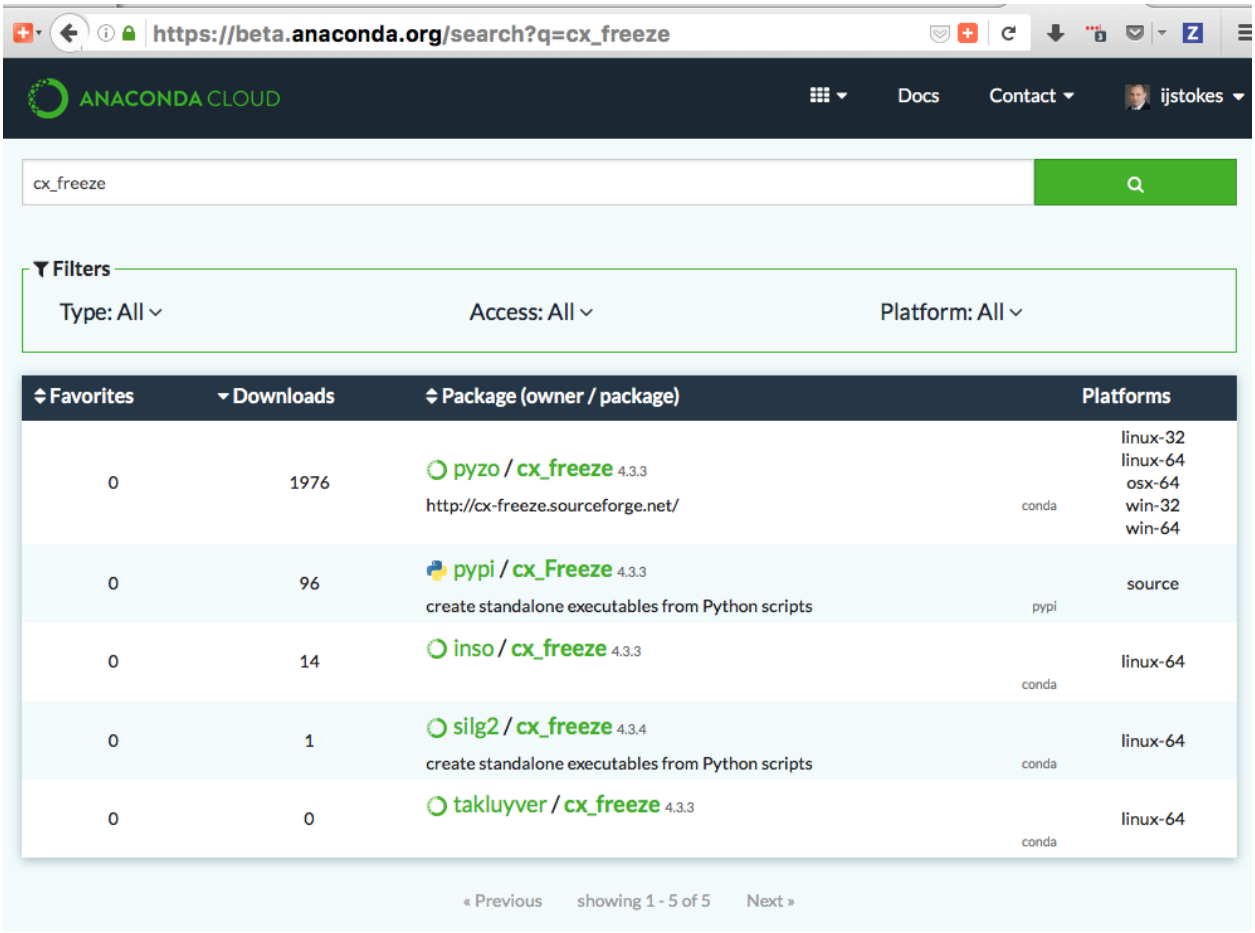
```
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata .........
Solving package specifications: .
Error: Package missing in current osx-64 channels:
- cx_freeze

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:


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 »

```
$ 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
    ------------------------------------------------------------
                                           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`.

1.7.15 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.

1.7.16 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.

1.7.17 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.

1.7.18 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.

1.7.19 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, use the Anaconda Prompt shortcut in your Windows start menu. If you have an existing cmd.exe session that you'd like to activate conda in, run: `call <your anaconda/miniconda install location>\Scripts\activate base`.

1.7.20 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"
```

Home | *Concepts* | *Getting started* | *Installation* | *Configuration* | *Tasks* | *Additional resources*

Get started

- *See what conda is and what it does*.
- *Learn conda concepts and fundamentals*.
- *Create your first conda project in 20 minutes*.
- *View system requirements and installation directions*.

Dive deeper

- *Configure your conda files*.
- Follow the *tasks* to *manage conda environments*, *channels*, *packages*, and more.

Additional resources

- *Cheat sheet*.
- *Troubleshooting*.

CHAPTER

TWO

CONDA CONFIGURATION

```
# ##########################################################
# ##              Channel Configuration                   ##
# ##########################################################

# # channels (sequence: primitive)
# #   aliases: channel
# #   env var string delimiter: ','
# #   The list of conda channels to include for relevant operations.
# #
# channels:
#   - defaults

# # channel_alias (str)
# #   The prepended url location to associate with channel names.
# #
# channel_alias: https://conda.anaconda.org

# # 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 --overide-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.
# #
```

(continues on next page)

(continued from previous page)

```
# allowlist_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
# #   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.
```

(continues on next page)

(continued from previous page)

```
# #
# 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: 5

# # experimental (sequence: primitive)
# #   env var string delimiter: ','
# #   List of experimental features to enable.
# #
# experimental: []


# ######################################################
# ##            Basic Conda Configuration             ##
# ######################################################
```

(continues on next page)

(continued from previous page)

```
# # 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
# #   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
```

(continues on next page)

(continued from previous page)

```
# #   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

# # 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, or (2) a path
# #   to a directory containing certificates of trusted CA.
# #
```

(continues on next page)

(continued from previous page)

```
# 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
# #   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: []
```

(continues on next page)

(continued from previous page)

```
# # 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: []

# # pip_interop_enabled (bool)
# #   Allow the conda solver to interact with non-conda-installed python
# #   packages.
# #
# pip_interop_enabled: 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.
# #
# solver: classic


# ##########################################################
# ##  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
```

(continues on next page)

(continued from previous page)

```
# # 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.
# #
# 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.
```

(continues on next page)

(continued from previous page)

```
# #
# 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

# # 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.
# #
# 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
```

(continues on next page)

(continued from previous page)

```
# #   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:

# # auto_activate_base (bool)
# #   Automatically activate the base environment during shell
# #   initialization.
# #
# auto_activate_base: true

# # 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
```

(continues on next page)

(continued from previous page)

```
# #   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

# # 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:

# # 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
```

(continues on next page)

(continued from previous page)

```
# #   higher this number, the longer the generation of the unsat hint will
# #   take. Defaults to 3.
# #
# 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
```

CHAPTER

THREE

CONDA PYTHON 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.

There are 3 supported public modules. We support:

1. import conda.cli.python_api
2. import conda.api
3. import conda.exports

The first 2 should have very long-term stability. The third is guaranteed to be stable throughout the lifetime of a feature release series--i.e. minor version number.

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.

3.1 conda.api.Solver

class DepsModifier(*value*)

Flags to enable alternate handling of dependencies.

NOT_SET = 'not_set'

NO_DEPS = 'no_deps'

ONLY_DEPS = 'only_deps'

class Solver(*prefix*, *channels*, *subdirs=()*, *specs_to_add=()*, *specs_to_remove=()*, *repodata_fn='repodata.json'*, *command=<conda.auxlib._Null object>*)

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()*

solve_final_state(*update_modifier=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*, *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]

solve_for_diff(*update_modifier=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*, *force_reinstall=<conda.auxlib._Null object>*, *should_retry_solve=False*)

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_for_transaction(*update_modifier=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*, *force_reinstall=<conda.auxlib._Null object>*, *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()*.

Returns

Return type UnlinkLinkTransaction

3.2 conda.cli.python_api

class Commands

CLEAN = 'clean'

CONFIG = 'config'

CREATE = 'create'

INFO = 'info'

INSTALL = 'install'

LIST = 'list'

NOTICES = 'notices'

REMOVE = 'remove'

RUN = 'run'

SEARCH = 'search'

UPDATE = 'update'

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=())
```

3.3 conda.api

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()*

solve_final_state(*update_modifier=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*)

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=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*, *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=<conda.auxlib._Null object>*, *deps_modifier=<conda.auxlib._Null object>*, *prune=<conda.auxlib._Null object>*, *ignore_pinned=<conda.auxlib._Null object>*, *force_remove=<conda.auxlib._Null object>*, *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()*.

Returns

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.

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]

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]

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.

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*

get(*package_ref*, *default=<conda.auxlib._Null object>*)

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

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

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]

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]

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.

`get(package_ref, default=<conda.auxlib._Null object>)`

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

`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

`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]

`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]

`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

CHAPTER

FOUR

COMMAND REFERENCE

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:

4.1 conda clean

Remove unused packages and caches.

Options:

```
usage: conda clean [-h] [-a] [-i] [-p] [-t] [-f] [-c [TEMPFILES ...]] [-l]
                   [-d] [--json] [-q] [-v] [-y]
```

4.1.1 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.

4.1.2 Output, Prompt, and Flow Control Options

-d, --dry-run Only display what would have been done.

--json Report all output as json. Suitable for using conda programmatically.

-q, --quiet Do not display progress bar.

-v, --verbose Can be used multiple times. Once for INFO, twice for DEBUG, three times for TRACE.

-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
```

4.2 conda compare

Compare packages between conda environments.

Options:

```
usage: conda compare [-h] [--json] [-v] [-q] [-n ENVIRONMENT | -p PATH] file
```

4.2.1 Positional Arguments

file Path to the environment file that is to be compared against.

4.2.2 Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

-v, --verbose Use once for info, twice for debug, three times for trace.

-q, --quiet Do not display progress bar.

4.2.3 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 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.

Options:

```
usage: conda config [-h] [--json] [-v] [-q] [--system | --env | --file FILE]
                    [--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]
```

4.3.1 Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

-v, --verbose Use once for info, twice for debug, three times for trace.

-q, --quiet Do not display progress bar.

4.3.2 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/envs/23.3.x/.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.

4.3.3 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*.

4.3.4 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
```

4.4 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 PREFIXoption.

Options:

```
usage: conda create [-h] [--clone ENV] (-n ENVIRONMENT | -p PATH) [-c CHANNEL]
                    [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [--strict-channel-priority] [--no-channel-priority]
                    [--no-deps | --only-deps] [--no-pin] [--copy] [-C] [-k]
                    [--offline] [-d] [--json] [-q] [-v] [-y] [--download-only]
                    [--show-channel-urls] [--file FILE]
                    [--no-default-packages]
                    [--solver {classic} | --experimental-solver {classic}]
                    [--dev]
                    [package_spec ...]
```

4.4.1 Positional Arguments

package_spec List of packages to install or update in the conda environment.

4.4.2 options

--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.

4.4.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

4.4.4 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.

4.4.5 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 Possible choices: classic

Choose which solver backend to use.

--experimental-solver Possible choices: classic

DEPRECATED. Please use '--solver' instead.

4.4.6 Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

4.4.7 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.

4.4.8 Output, Prompt, and Flow Control Options

-d, --dry-run Only display what would have been done.

--json Report all output as json. Suitable for using conda programmatically.

-q, --quiet Do not display progress bar.

-v, --verbose Can be used multiple times. Once for INFO, twice for DEBUG, three times for TRACE.

-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
```

4.5 conda info

Display information about current conda install.

Options:

```
usage: conda info [-h] [--json] [-v] [-q] [-a] [--base] [-e] [-s]
                  [--unsafe-channels]
```

4.5.1 options

-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.

4.5.2 Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
-v, --verbose	Use once for info, twice for debug, three times for trace.
-q, --quiet	Do not display progress bar.

4.6 conda init

Initialize conda for shell interaction.

Options:

```
usage: conda init [-h] [--all] [--user] [--no-user] [--system] [--reverse]
                  [--json] [-v] [-q] [-d]
                  [SHELLS ...]
```

4.6.1 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']

4.6.2 options

--all	Initialize all currently available shells.
-d, --dry-run	Only display what would have been done.

4.6.3 setup type

--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.

4.6.4 Output, Prompt, and Flow Control Options

--json	Report all output as json. Suitable for using conda programmatically.
-v, --verbose	Use once for info, twice for debug, three times for trace.
-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.

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.

4.7 conda install

Installs 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.

Options:

```
usage: conda install [-h] [--revision REVISION] [-n ENVIRONMENT | -p PATH]
                     [-c CHANNEL] [--use-local] [--override-channels]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                     [--strict-channel-priority] [--no-channel-priority]
                     [--no-deps | --only-deps] [--no-pin] [--copy] [-C] [-k]
                     [--offline] [-d] [--json] [-q] [-v] [-y]
                     [--download-only] [--show-channel-urls] [--file FILE]
                     [--solver {classic} | --experimental-solver {classic}]
                     [--force-reinstall]
                     [--freeze-installed | --update-deps | -S | --update-all | --update-
↪specs]
                     [-m] [--clobber] [--dev]
                     [package_spec ...]
```

4.7.1 Positional Arguments

package_spec List of packages to install or update in the conda environment.

4.7.2 options

--revision Revert to the specified REVISION.

--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.

4.7.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

4.7.4 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.

4.7.5 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 Possible choices: classic

Choose which solver backend to use.

--experimental-solver Possible choices: classic

DEPRECATED. Please use '--solver' instead.

--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 info --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.

4.7.6 Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

-m, --mkdir Create the environment directory, if necessary.

--clobber Allow clobbering (i.e. overwriting) of overlapping file paths within packages and suppress related warnings.

4.7.7 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.

4.7.8 Output, Prompt, and Flow Control Options

-d, --dry-run Only display what would have been done.

--json Report all output as json. Suitable for using conda programmatically.

-q, --quiet Do not display progress bar.

-v, --verbose Can be used multiple times. Once for INFO, twice for DEBUG, three times for TRACE.

-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=3.10
```

4.8 conda list

List installed packages in a conda environment.

Options:

```
usage: conda list [-h] [-n ENVIRONMENT | -p PATH] [--json] [-v] [-q]
                  [--show-channel-urls] [-c] [-f] [--explicit] [--md5] [-e]
                  [-r] [--no-pip]
                  [regex]
```

4.8.1 Positional Arguments

regex List only packages matching this regular expression.

4.8.2 options

--show-channel-urls Show channel urls. Overrides the value given by *conda config --show show_channel_urls*.

-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.

-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.

4.8.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

4.8.4 Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

-v, --verbose Use once for info, twice for debug, three times for trace.

-q, --quiet Do not display progress bar.

Examples:

List all packages in the current environment:

```
conda list
```

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
```

4.9 conda notices

Retrieves 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.

Options:

```
usage: conda notices [-h] [-c CHANNEL] [--use-local] [--override-channels]
                     [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
```

4.9.1 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.

Examples:

conda notices

conda notices -c defaults

4.10 conda package

Low-level conda package utility. (EXPERIMENTAL)

Options:

```
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]
```

4.10.1 options

-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.

4.10.2 Target Environment Specification

-n, --name	Name of environment.
-p, --prefix	Full path to environment location (i.e. prefix).

4.11 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.

Options:

```
usage: conda remove [-h] [-n ENVIRONMENT | -p PATH] [-c CHANNEL] [--use-local]
                    [--override-channels] [--repodata-fn REPODATA_FNS]
                    [--experimental {jlap,lock}] [--all] [--features]
                    [--force-remove] [--no-pin]
                    [--solver {classic} | --experimental-solver {classic}]
                    [-C] [-k] [--offline] [-d] [--json] [-q] [-v] [-y] [--dev]
                    [package_name ...]
```

4.11.1 Positional Arguments

package_name Package names to remove from the environment.

4.11.2 options

--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.

4.11.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

4.11.4 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.

4.11.5 Solver Mode Modifiers

--all Remove all packages, i.e., the entire environment.

--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 Possible choices: classic

Choose which solver backend to use.

--experimental-solver Possible choices: classic

DEPRECATED. Please use '--solver' instead.

4.11.6 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.

4.11.7 Output, Prompt, and Flow Control Options

-d, --dry-run Only display what would have been done.

--json Report all output as json. Suitable for using conda programmatically.

-q, --quiet Do not display progress bar.

-v, --verbose Can be used multiple times. Once for INFO, twice for DEBUG, three times for TRACE.

-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 environemnt 'myenv':

```
conda remove -n myenv scipy curl wheel
```

Remove all packages from environment *myenv* and the environment itself:

```
conda remove -n myenv --all
```

4.12 conda rename

Renames 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.

Options:

```
usage: conda rename [-h] [-n ENVIRONMENT | -p PATH] [--force] [-d] destination
```

4.12.1 Positional Arguments

destination New name for the conda environment.

4.12.2 options

--force Force rename of an environment.

-d, --dry-run Only display what would have been done by the current command, arguments, and other flags.

4.12.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

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
```

4.13 conda run

Run an executable in a conda environment.

Options:

```
usage: conda run [-h] [-n ENVIRONMENT | -p PATH] [-v] [--dev]
                 [--debug-wrapper-scripts] [--cwd CWD] [--no-capture-output]
                 ...
```

4.13.1 Positional Arguments

executable_call Executable name, with additional arguments to be passed to the executable on invocation.

4.13.2 options

-v, --verbose Use once for info, twice for debug, three times for trace.

--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).

4.13.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

Example usage:

```
$ conda create -y -n my-python-env python=3
$ conda run -n my-python-env python --version
```

4.14 conda search

Search for packages and display associated information.The input is a MatchSpec, a query language for conda packages.
See examples below.

Options:

```
usage: conda search [-h] [--envs] [-i] [--subdir SUBDIR] [-c CHANNEL]
                    [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--experimental {jlap,lock}]
                    [-C] [-k] [--offline] [--json] [-v] [-q]
```

4.14.1 options

--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.

4.14.2 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.

4.14.3 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.

4.14.4 Output, Prompt, and Flow Control Options

--json Report all output as json. Suitable for using conda programmatically.

-v, --verbose Use once for info, twice for debug, three times for trace.

-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]'
```

4.15 conda update

Updates 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.

Options:

```
usage: conda update [-h] [-n ENVIRONMENT | -p PATH] [-c CHANNEL] [--use-local]
                    [--override-channels] [--repodata-fn REPODATA_FNS]
                    [--experimental {jlap,lock}] [--strict-channel-priority]
                    [--no-channel-priority] [--no-deps | --only-deps]
                    [--no-pin] [--copy] [-C] [-k] [--offline] [-d] [--json]
                    [-q] [-v] [-y] [--download-only] [--show-channel-urls]
                    [--file FILE]
                    [--solver {classic} | --experimental-solver {classic}]
                    [--force-reinstall]
                    [--freeze-installed | --update-deps | -S | --update-all | --update-
↪specs]
                    [--clobber]
                    [package_spec ...]
```

4.15.1 Positional Arguments

package_spec List of packages to install or update in the conda environment.

4.15.2 options

--file Read package versions from the given file. Repeated file specifications can be passed (e.g. --file=file1 --file=file2).

4.15.3 Target Environment Specification

-n, --name Name of environment.

-p, --prefix Full path to environment location (i.e. prefix).

4.15.4 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.

4.15.5 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 Possible choices: classic

Choose which solver backend to use.

--experimental-solver Possible choices: classic

DEPRECATED. Please use '--solver' instead.

--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 info --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.

4.15.6 Package Linking and Install-time Options

--copy Install all packages using copies instead of hard- or soft-linking.

--clobber Allow clobbering of overlapping file paths within packages, and suppress related warnings.

4.15.7 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.

4.15.8 Output, Prompt, and Flow Control Options

-d, --dry-run Only display what would have been done.

--json Report all output as json. Suitable for using conda programmatically.

-q, --quiet Do not display progress bar.

-v, --verbose Can be used multiple times. Once for INFO, twice for DEBUG, three times for TRACE.

-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
```

4.16 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.

CHAPTER

FIVE

GLOSSARY

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".

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. 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.

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*.

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.

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.

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."

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.

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`

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.

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 or Anaconda Prompt using conda commands
- https://repo.anaconda.com/pkgs/

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.

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*.

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".

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.

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.

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.

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*.

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.

CHAPTER

SIX

DEVELOPER GUIDE

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 23.3.1.post2+bdcba5dd0 (Jul 17, 2023).

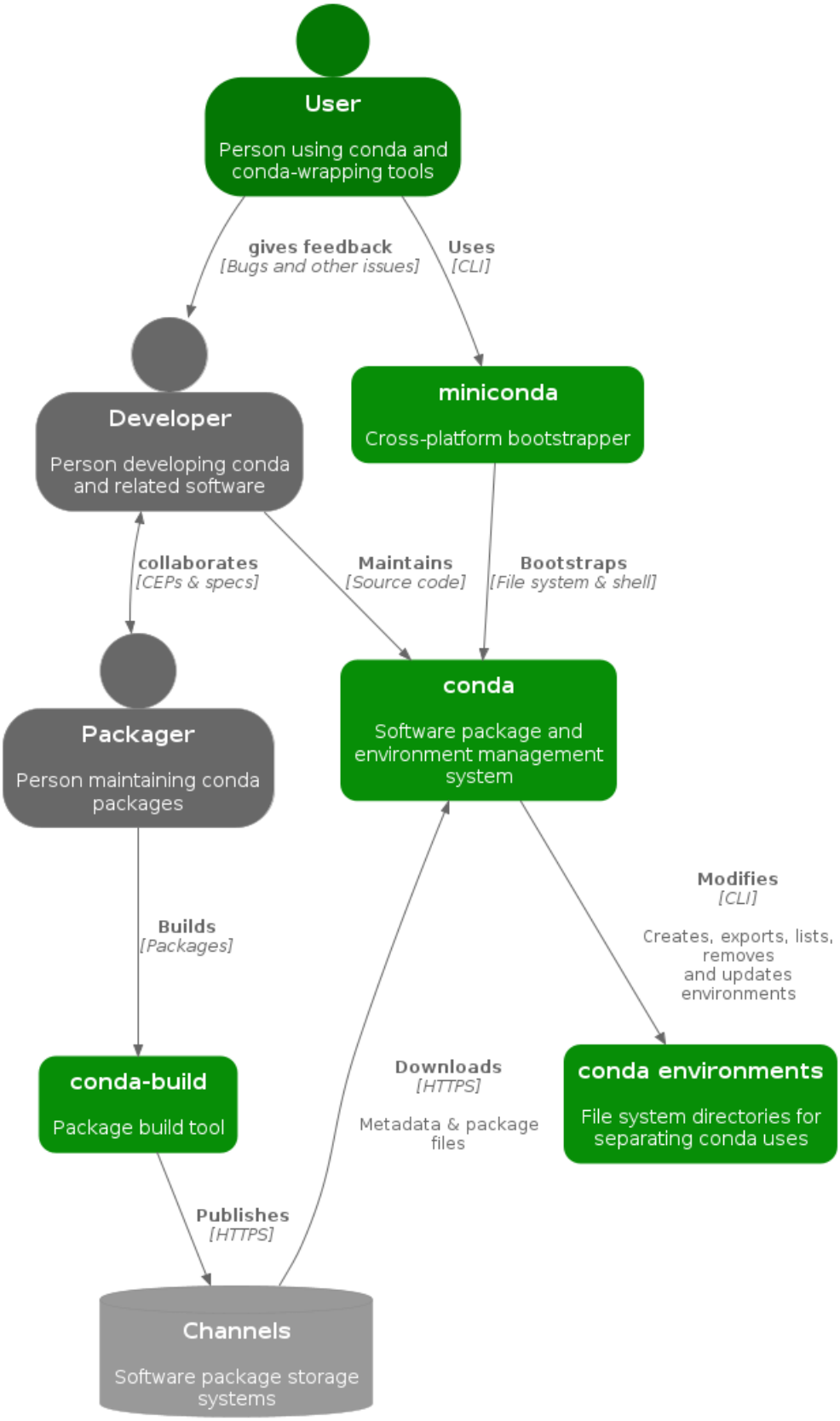
C4 stands for the for levels:

1. *Context*
2. *Container*
3. *Component*
4. *Code*

6.1.1 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

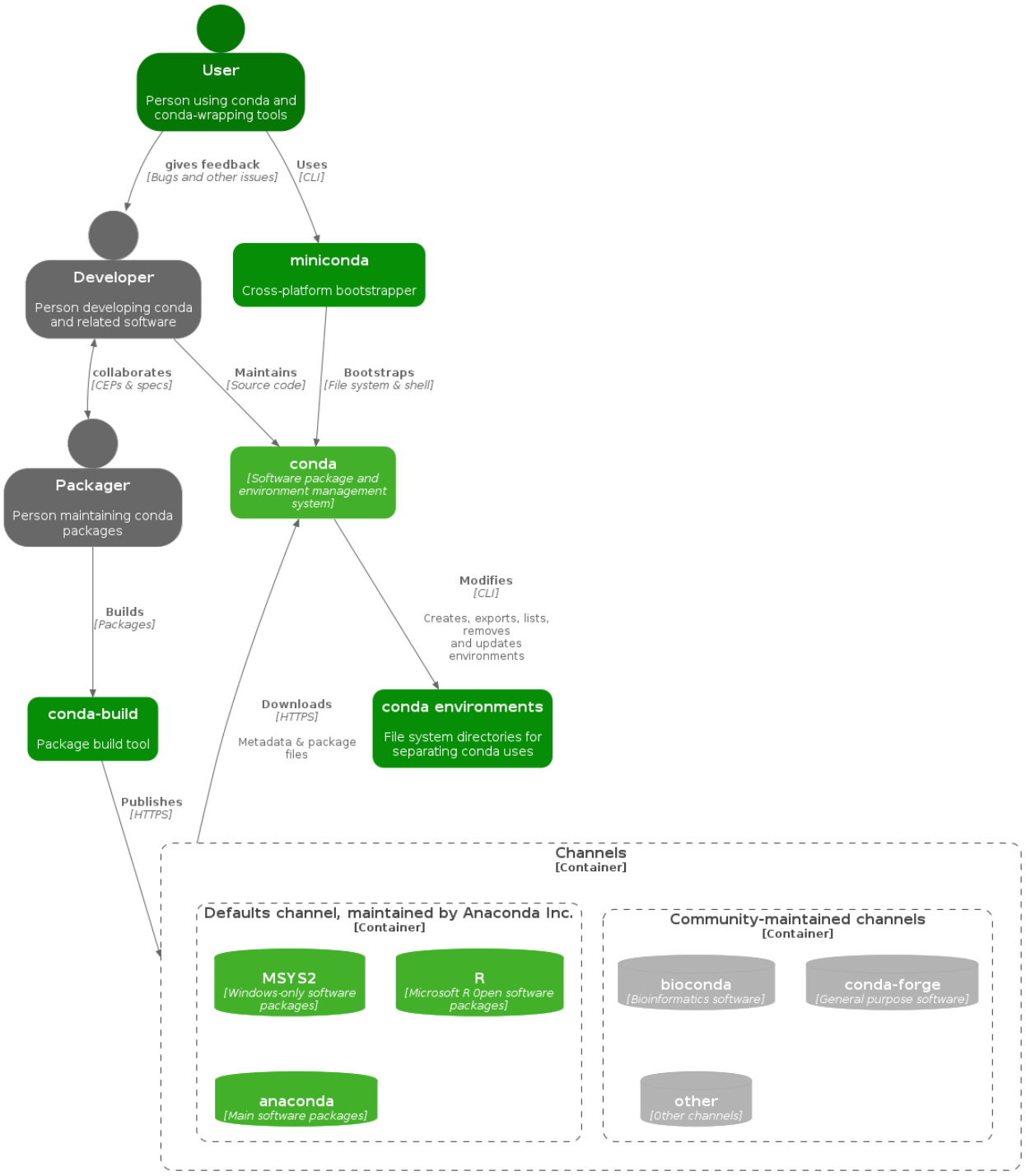
6.1.2 Level 2: Container

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.

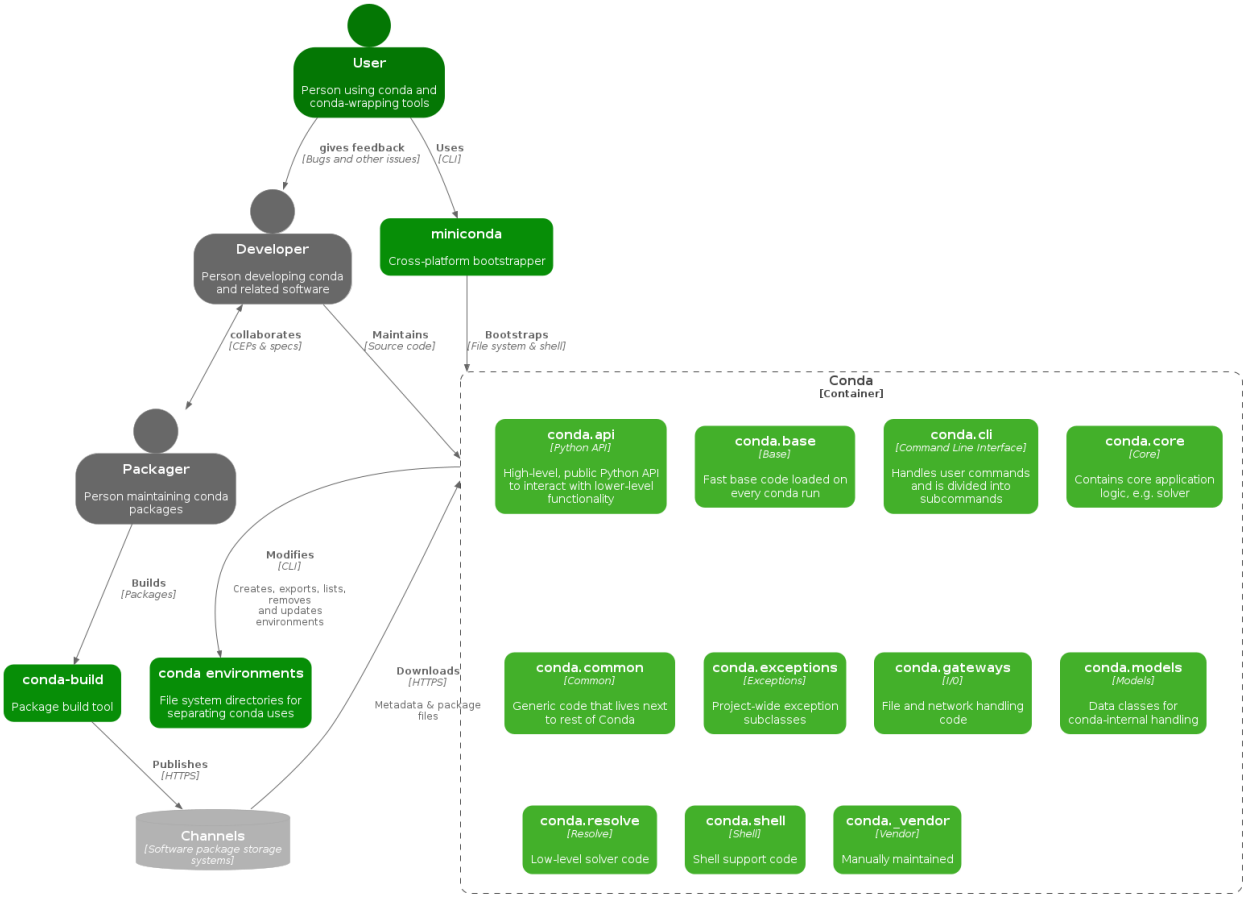
Channels

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]

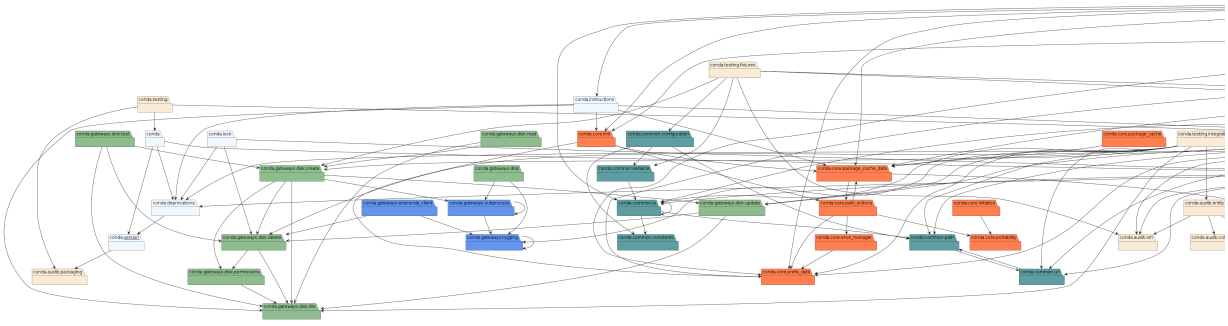
Conda

The following diagram focuses on the conda container from the level 1 diagram.



6.1.3 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.

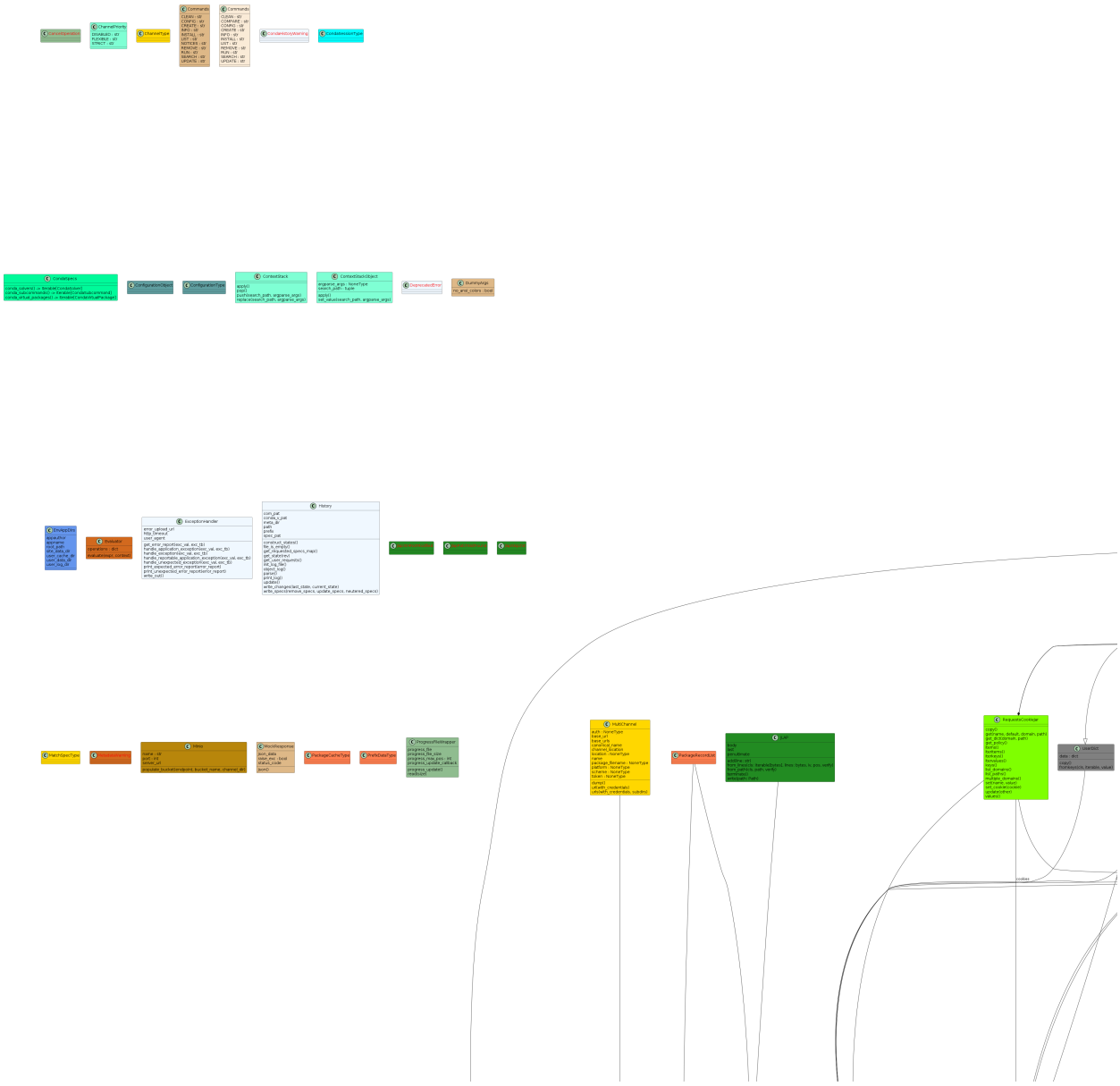


More information about how to interpret this diagram can be found in the C4 model documentation about the Component diagram.

6.1.4 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.



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

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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

6.2.5 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 formmating 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.

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. `3.10`.
   - `CONDA_DOCKER_DEFAULT_CHANNEL`: either `defaults` or `conda-forge`

   For example, if you need a conda-forge based 3.10 image:

   Bash (macOS, Linux, Windows)

```
$ CONDA_DOCKER_PYTHON=3.10 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=3.10 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose␣
↪build --no-cache --build-arg python_version=3.10 --build-arg default_
↪channel=conda-forge
$ CONDA_DOCKER_PYTHON=3.10 CONDA_DOCKER_DEFAULT_CHANNEL=conda-forge docker compose␣
↪run interactive
```

   cmd.exe (Windows)

```
> set CONDA_DOCKER_PYTHON=3.10
> 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.

6.3.1 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.

6.3.2 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.

6.4 Deep dives

This section contains a series of deep dives into particularly complex parts of conda.

6.4.1 `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

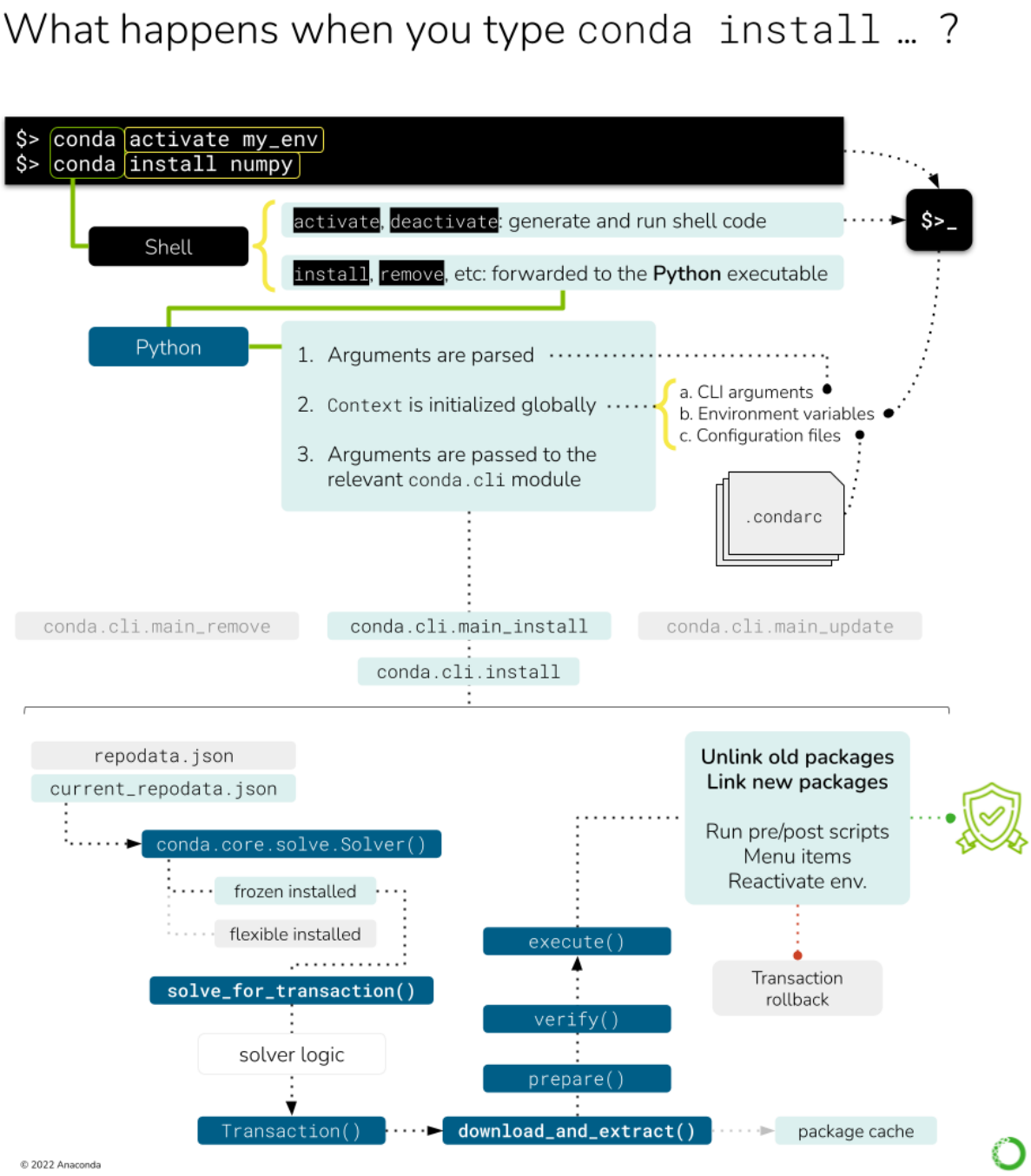


Fig. 1: This figure shows the different processes and objects involved in handling a simple `conda install` command.

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 *Conda 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 "conda 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 conda 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 solver logic: 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`: `LinkUnlinkTransaction`. It only accepts one input parameter: a list of `PrefixSetup` objects, which are just `namedtuple` objects with the followiing 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 `PrefixActionGroup` namedtuple. 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
      CreateNonadminAction
      CreatePythonEntryPointAction
      CreatePrefixRecordAction
      UpdateHistoryAction
    RemoveFromPrefixPathAction
```

(continues on next page)

(continued from previous page)

```
      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 `PrefixActionGroup`. 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.

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

6.4.2 `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` enviroment 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_
↪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'
```

6.4.3 `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.

6.4.4 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 /dev-guide/techspec-solver-state.

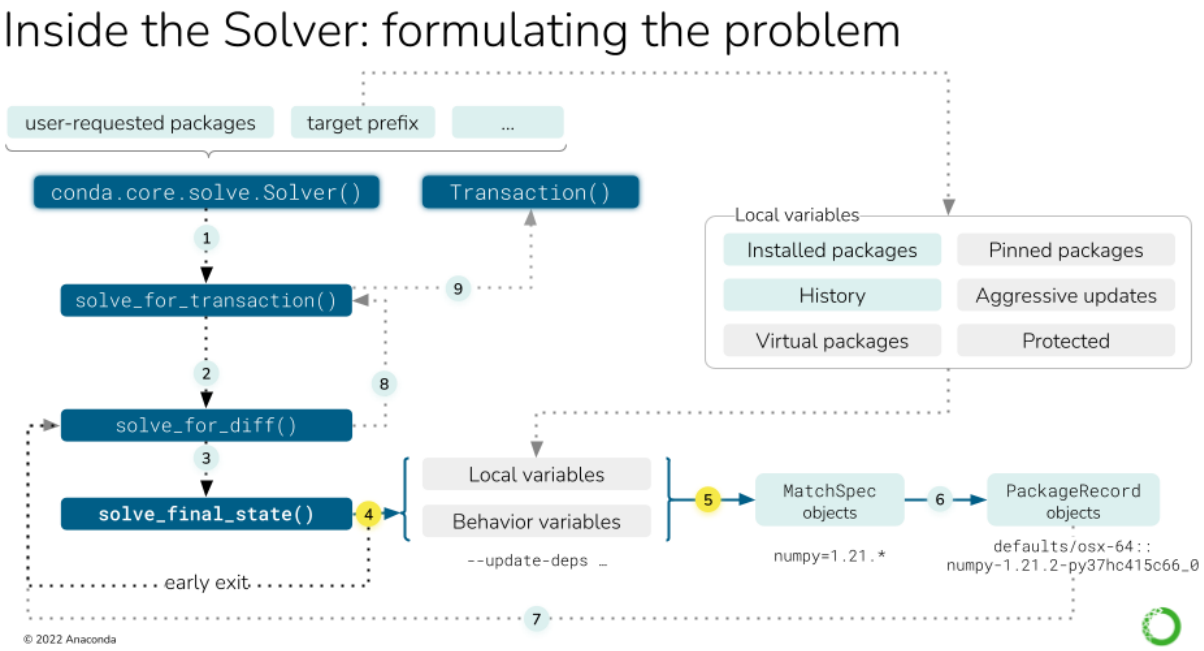


Fig. 2: Local variables affect the solving process explicitly and implicitly. As seen in *in the conda install deep dive*, the main actor is the `conda.core.solve.Solver` class. Before invoking the SAT solver, we can describe nine steps:

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*.

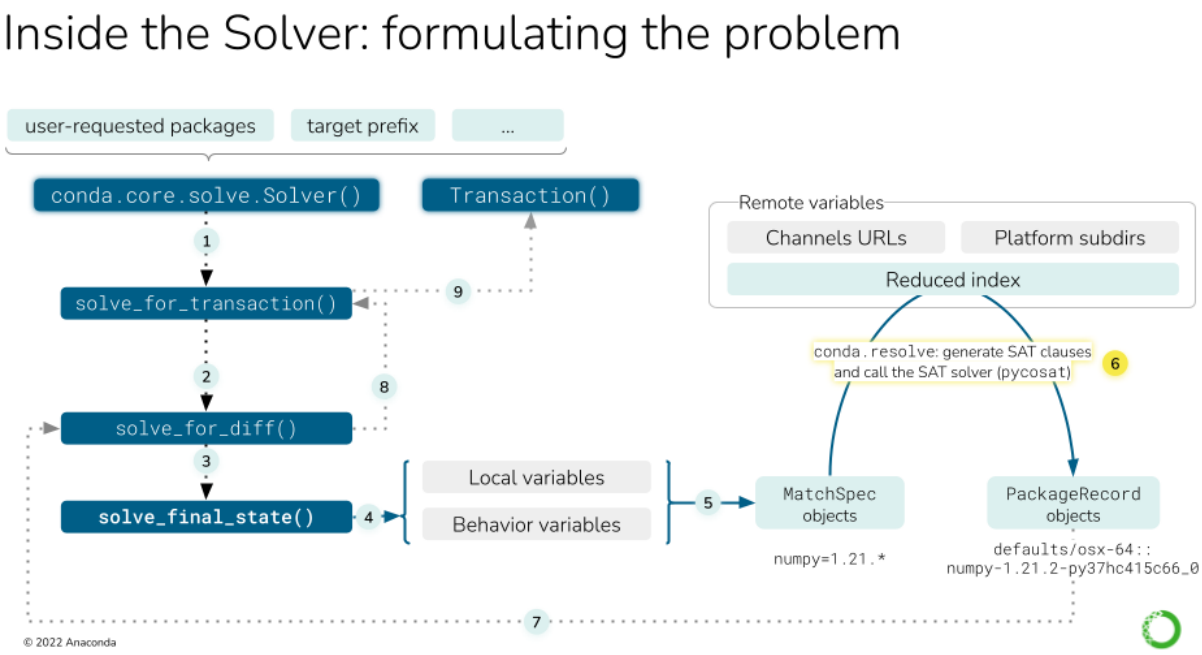


Fig. 3: The remote variables in a solve refer to, essentially, the package index (channels). This figure describes nine steps, focusing on 6-9. For steps 1-5, see *the previous figure*.

1. 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.
2. 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.
3. `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).
4. `solve_for_transaction` takes the diff and some more metadata in the instance to generate the `Transaction` object.

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).
- 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()`

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!

6.5 Writing Tests

This section contains a series of guides and guidelines for writing tests in the `conda` repository.

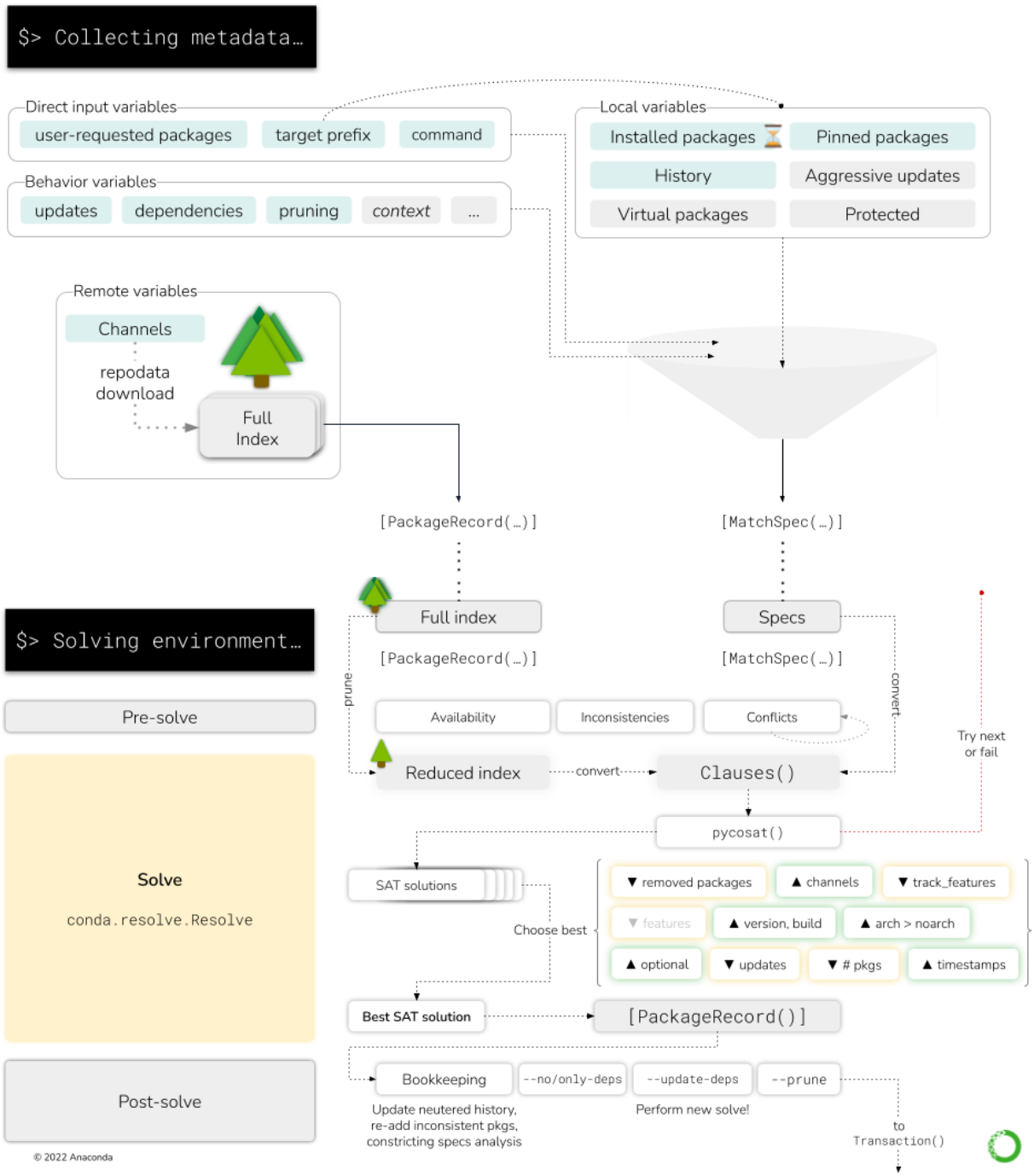


Fig. 4: Here you can see how the high level Solver API interacts with the low-level Resolve and Clauses objects.

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.

6.5.1 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.

Running CLI level tests

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:

```
import os.path
import json

from conda.testing.helpers import run_inprocess_conda_command as run

TEST_ENV_NAME_1 = "test-env-1"


def test_creates_new_environment():
    out, err, exit_code = run(f"conda create -n {TEST_ENV_NAME_1} -y")

    assert "conda activate test" in out  # ensure activation message is present
    assert err == ""  # no error messages
    assert exit_code == 0  # successful exit code

    # Perform a separate verification that everything works using the "conda env list"␣
↪command
    out, err, exit_code = run("conda env list --json")
    json_out = json.loads(out)
    env_names = {os.path.basename(path) for path in json_out.get("envs", tuple())}

    assert TEST_ENV_NAME_1 in env_names

    out, err, exit_code = run(f"conda remove --all -n {TEST_ENV_NAME_1}")

    assert err == ""
    assert exit_code == 0
```

Let's break down exactly what is going on in the code snippet above:

First, we import a function called `run_inprocess_conda_command` (aliased to `run` here) that allows us to run a command using the current running process. This ends up being much more efficient and quicker than running this test as a subprocess.

In the test itself, we first use our `run` function to create a new environment. 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 successfully ran.

The second part of the test again uses the `run` command 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 all environments currently 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. It is important to remember to remove anything you create, as it will be present when other tests are run.

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:

```
# Writing a test for `conda env list`

import os.path
import json

import pytest

from conda.testing.helpers import run_inprocess_conda_command as run

TEST_ENV_NAME_1 = "test-env-1"


@pytest.fixture()
def env_one():
    out, err, exit_code = run(f"conda create -n {TEST_ENV_NAME_1} -y")

    assert exit_code == 0

    yield

    out, err, exit_code = run(f"conda remove --all -n {TEST_ENV_NAME_1}")

    assert exit_code == 0


def test_env_list_finds_existing_environment(env_one):
    # Because we're using fixtures, we can immediately run the `conda env list` command
    # and our test assertions
    out, err, exit_code = run("conda env list --json")
    json_out = json.loads(out)
    env_names = {os.path.basename(path) for path in json_out.get("envs", tuple())}

    assert TEST_ENV_NAME_1 in env_names
```

(continues on next page)

(continued from previous page)

```
    assert err == ""
    assert exit_code == 0
```

In the fixture named `env_one`, we first create a new environment in exactly the same way as we did in our previous test. We make an assertion to ensure that it ran correctly and yield to mark the end of the setup. In the teardown section after the `yield` statement, we run the `conda remove` command and also make an assertion to determine it ran correctly.

This fixture will be run using the default scope in `pytest`, which is `function`. This means that the setup and teardown will be run before and after each test. 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.

6.5.2 General Guidelines

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,))

        # 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...
```

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.

6.6.1 Functions, Methods, and Properties

The simplest use case is for deprecating any function, method, or property:

Listing 1: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3")
def bar():
    ...
```

Listing 2: 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 3: Example file, `foo.py`.

```
from conda.deprecations import deprecated


@deprecated("23.9", "24.3", addendum="Use `qux` instead.")
```

(continues on next page)

(continued from previous page)

```
def bar():
    ...
```

Listing 4: Example invocation.

```
>>> import foo
>>> foo.bar()
<stdin>:1: PendingDeprecationWarning: foo.bar is pending deprecation and will be removed␣
↪in 24.3. Use `qux` instead.
```

6.6.2 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 5: Example file, `foo.py`.

```
from conda.deprecations import deprecated


# prior implementation
# def bar(is_true=True):
#     ...


@deprecated.argument("23.9", "24.3", "is_true")
def bar():
    ...
```

Listing 6: 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 7: Example file, `foo.py`.

```
from conda.deprecations import deprecated


# prior implementation
# def bar(is_true=True):
```

(continues on next page)

(continued from previous page)

```
#     ...


@deprecated.argument("23.9", "24.3", "is_true", rename="enabled")
def bar(enabled=True):
    ...
```

Listing 8: 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.
```

6.6.3 Constant

We also offer a way to deprecate global variables or constants:

Listing 9: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.constant("23.9", "24.3", "ULTIMATE_CONSTANT", 42)
```

Listing 10: Example invocation.

```
>>> import foo
>>> foo.ULTIMATE_CONSTANT
<stdin>:1: PendingDeprecationWarning: foo.ULTIMATE_CONSTANT is pending deprecation and␣
↪will be removed in 24.3.
```

Note: Constants deprecation relies on the module's `__getattr__` introduced in PEP-562.

6.6.4 Module

Entire modules can be also be deprecated:

Listing 11: Example file, `foo.py`.

```
from conda.deprecations import deprecated

deprecated.module("23.9", "24.3")
```

Listing 12: Example invocation.

```
>>> import foo
<stdin>:1: PendingDeprecationWarning: foo is pending deprecation and will be removed in␣
↪24.3.
```

6.6.5 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 13: 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 14: Example invocation.

```
>>> import foo
>>> foo.bar(...)
<stdin>:1: PendingDeprecationWarning: The <TOPIC> is pending deprecation and will be␣
↪removed in 24.3.
```

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!

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.

6.8.1 An overview of `pluggy`

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.

At its core, creating and implementing custom plugins with the use of `pluggy` can be broken down into two parts:

1. Define the hooks you want to register
2. Register your plugin under the `conda` entrypoint namespace

If you would like more information about `pluggy`, please refer to their documentation for a full description of its features.

6.8.2 Basic hook and entry point examples

Hook

Below is an example of a very basic plugin "hook":

Listing 15: my_plugin.py

```
import conda.plugins


@conda.plugins.hookimpl
def conda_subcommands():
    ...
```

Packaging / entry point namespace

The `pyproject.toml` file shown below is an example of a way to define and build a package out of the custom plugin hook shown above:

Listing 16: 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"]
```

(continues on next page)

(continued from previous page)

```
[project.entry-points."conda"]
my-conda-plugin = "my_plugin"
```

The `setup.py` file below is an alternative to the `pyproject.toml` file shown above; its main difference is the `entry_points` argument that is provided to the `setup()` function:

Listing 17: setup.py

```
from setuptools import setup

setup(
    name="my-conda-plugin",
    install_requires="conda",
    entry_points={"conda": ["my-conda-plugin = my_plugin"]},
    py_modules=["my_plugin"],
)
```

6.8.3 A note on licensing

When licensing plugins, we recommend using licenses such as BSD-3, MIT, and Apache License 2.0. Some `import` statements may possibly require the GPLv3 license, which ensures that the software being licensed is open source.

Ultimately, the authors of the plugins can decide which license is best for their particular use case. Be sure to credit the original author of the plugin, and keep in mind that licenses can be altered depending on the situation.

For more information on which license to use for your custom plugin, please reference the "Choose an Open Source License" site.

6.8.4 API reference

hookimpl
Conda plugin hook implementation marker.

Manager

The conda plugin manger extends pluggy's default plugin manager with a number of conda-specific features.

class CondaPluginManager(*project_name: Optional[str] = None*, **args*, ***kwargs*)
The conda plugin manager to implement behavior additional to pluggy's default plugin manager.

get_cached_solver_backend = None
Cached version of the *get_solver_backend()* method.

get_hook_results(*name: str*) → list
Return results of the plugin hooks with the given name and raise an error if there is an conflict.

get_solver_backend(*name: Optional[str] = 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.

load_entrypoints(*group: str*, *name: Optional[str] = None*) → int
Load modules from querying the specified setuptools `group`. :param str group: Entry point group to load plugins. :param str name: If given, loads only plugins with the given `name`. :rtype: int :return: The number of plugins loaded by this call.

load_plugins(**plugins*) → list[str]
Load the provided list of plugins and fail gracefully on error. The provided list plugins can either be classes or modules with `hook_impl`.

get_plugin_manager() → *conda.plugins.manager.CondaPluginManager*
Get a cached version of the *CondaPluginManager* instance, with the built-in and the entrypoints provided plugins loaded.

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(*name: str*, *backend: type[Solver]*)
A conda solver.

Parameters

- **name** -- Solver name (e.g., `custom-solver`).
- **backend** -- Type that will be instantiated as the solver backend.

backend: type[*conda.core.solve.Solver*]
Alias for field number 1

name: str
Alias for field number 0

conda_solvers(*self*) → collections.abc.Iterable[*conda.plugins.types.CondaSolver*]
Register solvers in conda.

Returns An iterable of solvers entries.

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():
    yield plugins.CondaSolver(
```

(continues on next page)

(continued from previous page)

```
        name="verbose-classic",
        backend=VerboseSolver,
    )
```

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(*name: str*, *summary: str*, *action: Callable[[list[str]], int | None]*)

A conda subcommand.

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.

action: Callable[[list[str]], int | None]

Alias for field number 2

name: str

Alias for field number 0

summary: str

Alias for field number 1

conda_subcommands(*self*) → collections.abc.Iterable[*conda.plugins.types.CondaSubcommand*]

Register external subcommands in conda.

Returns An iterable of subcommand entries.

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,
    )
```

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(*name: str*, *version: str | None*, *build: str | None*)

A conda virtual package.

Parameters

- **name** -- Virtual package name (e.g., `my_custom_os`).
- **version** -- Virtual package version (e.g., `1.2.3`).
- **version** -- Virtual package build string (e.g., `x86_64`).

build: str | None

Alias for field number 2

name: str

Alias for field number 0

version: str | None

Alias for field number 1

conda_virtual_packages(*self*) → collections.abc.Iterable[*conda.plugins.types.CondaVirtualPackage*]

Register virtual packages in Conda.

Returns An iterable of virtual package entries.

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",
    )
```

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.

6.9.1 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:

1. `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.

6.9.2 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.

6.9.3 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

6.9.4 Processing specs for `conda remove`

CHAPTER

SEVEN

RELEASE NOTES

This information is drawn from the GitHub conda project changelog: https://github.com/conda/conda/blob/main/CHANGELOG.md

7.1 23.3.1 (2023-03-28)

7.1.1 Enhancements

- Fix and re-enable binstar tests. Replace custom property caching with `functools.cached_property`. (#12495)

7.1.2 Bug fixes

- Restore default argument for `SubdirData` method used by `conda-index`. (#12513)
- Include `conda.gateways.repodata.jlap` submodule in package. (#12545)

7.1.3 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)

7.1.4 Contributors

- @conda-bot
- @dbast
- @dholth
- @jaimergp
- @kenodegard
- @ForgottenProgramme

7.2 23.3.0 (2023-03-14)

7.2.1 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)

7.2.2 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)

7.2.3 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)

7.2.4 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)

7.2.5 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)

7.2.6 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]

7.3 23.1.0 (2023-01-17)

7.3.1 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)

7.3.2 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)

7.3.3 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)

7.3.4 Contributors

- @beeankha
- @dholth
- @dariocurr made their first contribution in https://github.com/conda/conda/pull/12128
- @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]

7.4 22.11.1 (2022-12-06)

7.4.1 Bug fixes

- Restore default virtual package specs as in 22.9.0 (#12148)
  - re-add `__unix`/`__win` packages
  - restore `__archspec` version/build string composition

7.4.2 Other

- Skip test suite for non-code changes. (#12141)

7.4.3 Contributors

- @LtDan33
- @jezdez
- @kenodegard
- @mbargull
- @travishathaway

7.5 22.11.0 (2022-11-23)

7.5.1 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)

7.5.2 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)

7.5.3 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)

7.5.4 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)

7.5.5 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)

7.5.6 Contributors

- @arq0017 made their first contribution in https://github.com/conda/conda/pull/11810
- @beeankha
- @conda-bot
- @dbast
- @dholth
- @erykoff
- @consideRatio 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

7.6 22.9.0 (2022-09-14)

7.6.1 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.

7.6.2 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)

7.6.3 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)

7.6.4 Deprecations

- Mark `conda._vendor.toolz` as pending deprecation. (#11704)
- Removes vendored version of urllib3. (#11705)

7.6.5 Docs

- Added conda capitalization standards to CONTRIBUTING file. (#11712)

7.6.6 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)

7.6.7 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]

7.7 4.14.0 (2022-08-02)

7.7.1 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)

7.7.2 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)

7.7.3 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)

7.7.4 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)

7.7.5 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)

7.7.6 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]

7.8 4.13.0 (2022-05-19)

7.8.1 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)

7.8.2 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)

7.8.3 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)

7.8.4 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)

7.8.5 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)

7.8.6 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]

7.9 4.12.0 (2022-03-08)

7.9.1 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)

7.9.2 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)

7.9.3 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)

7.9.4 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)

7.9.5 Contributors

- @opoplawski
- @FaustinCarter
- @jaimergp
- @rhoule-anaconda
- @jezdez
- @hajapy
- @erykoff
- @uwuvalon
- @kenodegard
- @manics
- @NaincyKumariKnoldus
- @autotmp
- @yuvipanda
- @astrojuanlu
- @marcelotrevisani

7.10 4.11.0 (2021-11-22)

7.10.1 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).

7.10.2 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.

7.10.3 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.

7.10.4 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.

7.10.5 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

7.11 4.10.3 (2021-06-29)

7.11.1 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)

7.12 4.10.2 (2021-06-25)

7.12.1 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)

7.12.2 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)

7.12.3 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)

7.12.4 Contributors

- @AlbertDeFusco
- @awwad
- @casperdcl
- @cgranade
- @chenghlee
- @ColemanTom
- @dan-hook
- @dbast
- @ericpre
- @HedgehogCode
- @jamesp
- @jezdez
- @johnhany97
- @lightmare
- @mattip
- @maxerbubba
- @mrakitin
- @stinos
- @thermokarst

7.13 4.10.1 (2021-04-12)

7.13.1 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)

7.13.2 Contributors

- @awwad
- @chenghlee
- @codepage949
- @niklasholm

7.14 4.10.0 (2021-03-30)

NOTE: This release formally drops support for Python 2.7 and Python < 3.6.

7.14.1 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)

7.14.2 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)

7.14.3 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)

7.14.4 Miscellaneous

- CI: drop Python 2.7 and add Python 3.9 (#10548)

7.14.5 Contributors

- @awwad
- @BastianZim
- @beenje
- @bgobbi
- @blubs
- @chenghlee
- @cjmartian
- @ericpre
- @erykoff
- @felker
- @giladmaya
- @jamesmyatt
- @mingwandroid
- @opoplawski
- @saadparwaiz1
- @saucoide

7.15 4.9.2 (2020-11-10)

7.15.1 Enhancements

- Use vendored `tqdm` in `conda.resolve` for better consistency (#10337)

7.15.2 Bug fixes

- Revert to previous naming scheme for repodata cache files when `use_only_tar_bz2` config option is false (#10350)

7.15.3 Docs

- Fix missing release notes (#10342)
- Fix permission errors when configuring deb repositories (#10347)

7.15.4 Contributors

- @chenghlee
- @csoja
- @dylanmorroll
- @sscherfke

7.16 4.9.1 (2020-10-26)

7.16.1 Enhancements

- Respect PEP 440 ~= "compatible release" clause (#10313)

7.16.2 Bug fixes

- Remove `preload_openssl` for Win32 (#10298)
- Add `if exist` to Windows registry hook (#10305)

7.16.3 Contributors

- @mingwandroid

7.17 4.9.0 (2020-10-19)

7.17.1 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)

7.17.2 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)

7.17.3 Docs

- Correctly state default `/AddToPath` option in Windows installer (#10179)
- Fix typos in `--repodata-fn` help text (#10279)

7.17.4 Miscellaneous

- Update CI infrastructure to use GitHub Actions (#10176, #10186, #10234)
- Update README badge to show GitHub Actions status (#10254)

7.17.5 Contributors

- @AlbertDeFusco
- @angloyna
- @bbodenmiller
- @casperdcl
- @chenghlee
- @chrisburr
- @cjmartian
- @dhirschfeld
- @ericpre
- @gabrielcnr
- @InfiniteChai

- @isuruf
- @jjhelmus
- @LorcanHamill
- @maresb
- @mingwandroid
- @mlline00
- @xhochy
- @ydmytryk

7.18 4.8.5 (2020-09-14)

7.18.1 Enhancements

- Add `osx-arm64` as a recognized platform (#10128, #10134)

7.18.2 Contributors

- @isuruf
- @jjhelmus

7.19 4.8.4 (2020-08-06)

7.19.1 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)

7.19.2 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)

7.19.3 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)

7.19.4 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)

7.19.5 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

7.20 4.8.3 (2020-03-13)

7.20.1 Docs

- Add release notes for 4.8.2 to docs (#9632)
- Fix typos in docs (#9637, #9643)
- Grammatical and formatting changes (#9647)

7.20.2 Bug fixes:

- Account for channel is specs (#9748)

7.20.3 Contributors

- @bernardoduarte
- @forrestwaters
- @jjhelmus
- @msarahan
- @rrigdon
- @timgates42

7.21 4.8.2 (2020-01-24)

7.21.1 Enhancements

- Solver messaging improvements (#9560)

7.21.2 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)

7.21.3 Bug fixes:

- change fish prompt only if changeps1 is true (#7000)
- make frozendict JSON serializable (#9539)
- Conda env create empty dir (#9543)

7.21.4 Contributors

- @msarahan
- @jjhelmus
- @rrigdon
- @soapy1
- @teake
- @csoja
- @kfranz

7.22 4.8.1 (2019-12-19)

7.22.1 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)

7.22.2 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)

7.22.3 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)

7.22.4 Contributors

- @AntoinePrv
- @brettcannon
- @bwildenhain
- @cjmartian
- @felker
- @forrestwaters
- @gilescope
- @isuruf
- @jeremyjliu
- @jjhelmus
- @jhultman
- @marcuscaisey
- @mbargull
- @mingwandroid
- @msarahan
- @okhoma
- @osamoylenko
- @rrigdon
- @ruleroftheuns
- @soapy1
- @tartansandal

7.23 4.8.0 (2019-11-04)

7.23.1 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)

7.23.2 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)

7.23.3 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)

7.23.4 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

7.24 4.7.12 (2019-09-12)

7.24.1 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)

7.24.2 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)

7.24.3 Contributors

- @beenje
- @Bezier89
- @cjmartian
- @forrestwaters
- @jjhelmus
- @martin-raden
- @msarahan
- @nganani
- @rrigdon
- @soapy1
- @WesRoach
- @zheaton

7.25 4.7.11 (2019-08-06)

7.25.1 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)

7.25.2 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)

7.25.3 Docs

- Document new package format better
- Document `conda init` command
- Document availability of RSS feed for CDN-backed channels that clone

7.25.4 Contributors

- @Bezier89
- @forrestwaters
- @hajapy
- @ihnorton
- @matthewwardrop
- @msarahan
- @rogererens
- @rrigdon
- @soapy1

7.26 4.7.10 (2019-07-19)

7.26.1 Bug fixes

- fix merging of specs
- fix bugs in building of chains in prefix graph

7.26.2 Contributors

- @msarahan

7.27 4.7.9 (2019-07-18)

7.27.1 Bug fixes

- fix Non records in comprehension
- fix potential keyerror in depth-first search
- fix PackageNotFound attribute error

7.27.2 Contributors

- @jjhelmus
- @msarahan

7.28 4.7.8 (2019-07-17)

7.28.1 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)

7.28.2 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)

7.28.3 Contributors

- @forrestwaters
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon
- @soapy1

7.29 4.7.7 (2019-07-12)

7.29.1 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)

7.29.2 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)

7.29.3 Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

7.30 4.7.6 (2019-07-11)

7.30.1 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)

7.30.2 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)

7.30.3 Contributors

- @forrestwaters
- @jjhelmus
- @mcopes73
- @msarahan
- @richardjgowers
- @rrigdon
- @soapy1
- @twinssbc

7.31 4.7.5 (2019-06-24)

7.31.1 Improvements

- improve wording in informational message when a particular `*_repodata.json` can't be found. No need for alarm. (#8808)

7.31.2 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)

7.31.3 Contributors

- @dhirschfeld
- @msarahan
- @rrigdon

7.32 4.7.4 (2019-06-19)

7.32.1 Improvements

- Revert to and improve the unsatisfiability determination from 4.7.2 that was reverted in 4.7.3. It's faster. (#8783)

7.32.2 Bug fixes

- fix tcsh/csh init scripts (#8792)

7.32.3 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.

7.32.4 Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @soapy1

7.33 4.7.3 (2019-06-14)

7.33.1 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)

7.33.2 Contributors

- @jjhelmus
- @msarahan
- @rrigdon
- @ScottEvtuch

7.34 4.7.2 (2019-06-10)

7.34.1 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)

7.34.2 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)

7.34.3 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)

7.34.4 Contributors

- @csosborn
- @jjhelmus
- @katietz
- @msarahan
- @rrigdon

7.35 4.7.1 (2019-05-30)

7.35.1 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)

7.35.2 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)

7.35.3 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.

7.35.4 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)

7.35.5 Contributors

- @alexhall
- @cjmartian
- @kalefranz
- @martinkou
- @msarahan
- @rrigdon
- @soapy1

7.36 4.7.0 (2019-05-17)

7.36.1 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 transacation verification and execution (#8670)

7.36.2 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)

7.36.3 Contributors

- @chrisburr
- @EternalPhane
- @jjhelmus
- @kalefranz
- @mbargull
- @msarahan
- @rrigdon
- @scopatz
- @seibert
- @soapy1
- @nehaljwani
- @nh3
- @teake
- @yuvalreches

7.37 4.6.14 (2019-04-17)

7.37.1 Bug fixes

- export extra function in powershell Conda.psm1 script (fixes anaconda powershell prompt) (#8570)

7.37.2 Contributors

- @msarahan

7.38 4.6.13 (2019-04-16)

7.38.1 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)

7.38.2 Contributors

- @dennispg
- @jjhelmus
- @jpgill86
- @mingwandroid
- @msarahan
- @noahp

7.39 4.6.12 (2019-04-10)

7.39.1 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)

7.39.2 Docs improvements

- Fix links in README (#8482)
- Changelogs for 4.6.10 and 4.6.11 (#8502)

7.39.3 Contributors

@Bezier89 @duncanmmacleod @ivigamberdiev @javabrett @jjhelmus @katietz @mingwandroid @msarahan @nehaljwani @rrigdon

7.40 4.6.11 (2019-04-04)

7.40.1 Bug fixes

- Remove sys.prefix from front of PATH in basic_posix (#8491)
- add import to fix conda.core.index.get_index (#8495)

7.40.2 Docs improvements

- Changelogs for 4.6.10

7.40.3 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

7.41 4.6.10 (2019-04-01)

7.41.1 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)

7.41.2 Docs improvements

- Changelogs for 4.6.8, 4.6.9

7.41.3 Contributors

- @duncanmmacleod
- @nehaljwani
- @ilango100
- @jjhelmus
- @mingwandroid
- @msarahan

- @rrigdon

7.42 4.6.9 (2019-03-29)

7.42.1 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)

7.42.2 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)

7.42.3 Docs improvements

- updated cheatsheet (#8402)
- updated color theme (#8403)

7.42.4 Contributors

- @blackgear
- @dhirschfeld
- @jakirkham
- @jjhelmus
- @katietz
- @mingwandroid
- @msarahan
- @nehaljwani
- @rrigdon
- @soapy1
- @spamlrot-tic

7.43 4.6.8 (2019-03-06)

7.43.1 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)

7.43.2 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)

7.43.3 Contributors

- @albertmichaelj
- @jjhelmus
- @matta9001
- @msarahan
- @rrigdon
- @soapy1
- @steffenvan

7.44 4.6.7 (2019-02-21)

7.44.1 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)

7.44.2 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)

7.44.3 Contributors

- @jjhelmus
- @MarckK
- @msarahan
- @rrigdon
- @samgd

7.45 4.6.6 (2019-02-18)

7.45.1 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)

7.45.2 Contributors

- @mingwandroid
- @msarahan

7.46 4.6.5 (2019-02-15)

7.46.1 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)

7.46.2 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan
- @rrigdon

7.47 4.6.4 (2019-02-13)

7.47.1 Improvements

- allow configuring location of instrumentation records (#7849)
- prepend conda-env pip commands with env activation to fix library loading (#8263)

7.47.2 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)

7.47.3 Deprecations/Breaking Changes

- remove experimental `featureless_minimization_disabled` feature flag (#8249)

7.47.4 Contributors

- @davemasino
- @geremih
- @jjhelmus
- @kalefranz
- @msarahan
- @minrk
- @nehaljwani
- @prusse-martin
- @rrigdon
- @soapy1

7.48 4.6.3 (2019-02-07)

7.48.1 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)

7.48.2 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)

7.48.3 Contributors

- @dhirschfeld
- @jjhelmus
- @kalefranz
- @msarahan
- @murrayreadccdc
- @nehaljwani
- @rrigdon
- @soapy1

7.49 4.6.2 (2019-01-29)

7.49.1 Improvements

- Documentation restructuring/improvements (#8139, #8143)
- rewrite rm_rf to use native system utilities and rename trash files (#8134)

7.49.2 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)

7.49.3 Contributors

- @jjhelmus
- @msarahan
- @nehaljwani
- @rrigdon

7.50 4.5.13 (2019-01-29)

7.50.1 Improvements

- document the allow_conda_downgrades configuration parameter (#8034)
- remove conda upgrade message (#8161)

7.50.2 Contributors

- @msarahan
- @nehaljwani

7.51 4.6.1 (2019-01-21)

7.51.1 Improvements

- optimizations in `get_reduced_index` (#8117, #8121, #8122)

7.51.2 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)

7.51.3 Contributors

- @jjhelmus
- @msarahan
- @pv

7.52 4.6.0 (2019-01-15)

7.52.1 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)

7.52.2 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)

7.52.3 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)

7.52.4 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)

7.52.5 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)

7.52.6 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

7.52.7 Contributors

- @cgranade
- @fabioz
- @geremih
- @goanpeca
- @jesse-
- @jjhelmus
- @kalefranz
- @makbigc
- @mandeep
- @mbargull
- @msarahan
- @nehaljwani
- @ohadravid
- @teake

7.53 4.5.12 (2018-12-10)

7.53.1 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)

7.53.2 Bug Fixes

- update integration tests for removal of 'features' key (#7726)
- fix conda.bat return code (#7944)
- ensure channel name is not NoneType (#8021)

7.53.3 Contributors

- @debionne
- @jjhelmus
- @kalefranz
- @msarahan
- @nehaljwani

7.54 4.5.11 (2018-08-21)

7.54.1 Improvements

- resolve #7672 compatibility with ruamel.yaml 0.15.54 (#7675)

7.54.2 Contributors

- @CJ-Wright
- @mbargull

7.55 4.5.10 (2018-08-13)

7.55.1 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)

7.55.2 Contributors

- @jesse-
- @nehaljwani

7.56 4.5.9 (2018-07-30)

7.56.1 Improvements

- resolve #7522 prevent conda from scheduling downgrades (#7598)
- allow skipping feature maximization in resolver (#7601)

7.56.2 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)

7.56.3 Contributors

- @kalefranz
- @loriab

7.57 4.5.8 (2018-07-10)

7.57.1 Bug Fixes

- fix #7524 should_bypass_proxies for requests 2.13.0 and earlier (#7525)

7.57.2 Contributors

- @kalefranz

7.58 4.5.7 (2018-07-09)

7.58.1 Improvements

- resolve #7423 add upgrade error for unsupported repodata_version (#7415)
- raise CondaUpgradeError for conda version downgrades on environments (#7517)

7.58.2 Bug Fixes

- fix #7505 temp directory for UnlinkLinkTransaction should be in target prefix (#7516)
- fix #7506 requests monkeypatch fallback for old requests versions (#7515)

7.58.3 Contributors

- @kalefranz
- @nehaljwani

7.59 4.5.6 (2018-07-06)

7.59.1 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)

7.59.2 Contributors

- @kalefranz

7.60 4.5.5 (2018-06-29)

7.60.1 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)

7.60.2 Contributors

- @goanpeca
- @kalefranz
- @mingwandroid
- @mbargull

7.61 4.5.4 (2018-05-14)

7.61.1 Improvements

- resolve #7189 progress bar improvement (#7191 via #7274)

7.61.2 Bug Fixes

- fix twofold tarball extraction, improve progress update (#7275)
- fix #7253 always respect copy LinkType (#7269)

7.61.3 Contributors

- @jakirkham
- @kalefranz
- @mbargull

7.62 4.5.3 (2018-05-07)

7.62.1 Bug Fixes

- fix #7240 conda's configuration context is not initialized in conda.exports (#7244)

7.63 4.5.2 (2018-04-27)

7.63.1 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)

7.64 4.5.1 (2018-04-13)

7.64.1 Improvements

- resolve #7075 add anaconda.org search message to PackagesNotFoundError (#7076)
- add CondaError details to auto-upload reports (#7060)

7.64.2 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)

7.65 4.5.0 (2018-03-20)

7.65.1 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.

7.65.2 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)

7.65.3 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)

7.65.4 API

- resolve #5610 add PrefixData, SubdirData, and PackageCacheData to conda/api.py (#6922)

7.65.5 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)

7.65.6 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)

7.66 4.4.11 (2018-02-23)

7.66.1 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)

7.66.2 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)

7.67 4.4.10 (2018-02-09)

7.67.1 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)

7.68 4.4.9 (2018-02-06)

7.68.1 Improvements

- resolve #6632 display package removal plan when deleting an env (#6801)

7.68.2 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)

7.69 4.4.8 (2018-01-25)

7.69.1 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)

7.69.2 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)

7.70 4.4.7 (2018-01-08)

7.70.1 Improvements

- resolve #6650 add upgrade message for unicode errors in python 2 (#6651)

7.70.2 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)

7.71 4.4.6 (2017-12-31)

7.71.1 Bug Fixes

- fix #6612 do not assume Anaconda Python on Windows nor Library\bin hack (#6615)
- recipe test improvements and associated bug fixes (#6614)

7.72 4.4.5 (2017-12-29)

7.72.1 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)

7.73 4.4.4 (2017-12-24)

7.73.1 Improvements

- add SUDO_ env vars to info reports (#6563)
- add additional information to the #6546 exception (#6551)

7.73.2 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)

7.74 4.4.3 (2017-12-22)

7.74.1 Improvements

- adjust error report message (#6534)

7.74.2 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)

7.74.3 Non-User-Facing Changes

- regression tests for #6512 (#6515)

7.75 4.4.2 (2017-12-22)

7.75.1 Deprecations/Breaking Changes

- resolve #6523 don't prune with --update-all (#6524)

7.75.2 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)

7.76 4.4.1 (2017-12-21)

7.76.1 Bug Fixes

- fix #6512 reactivate does not accept arguments (#6513)

7.77 4.4.0 (2017-12-20)

7.77.1 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.

7.77.2 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.

7.77.3 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)

7.77.4 API

- Add Solver from conda.core.solver with three methods to conda.api (4.4.0rc1) (#5838)

7.77.5 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)

7.77.6 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)

7.77.7 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)

7.78 4.3.34 (2018-02-09)

7.78.1 Bug Fixes

- fix #6833 improve feature mismatch metric (#6853)

7.79 4.3.33 (2018-01-24)

7.79.1 Bug Fixes

- fix #6718 broken 'conda install --rev' (#6719)
- fix #6765 adjust the feature score assigned to packages not installed (#6766)

7.80 4.3.32 (2018-01-10)

7.80.1 Improvements

- resolve #6711 fall back to copy/unlink for EINVAL, EXDEV rename failures (#6712)

7.80.2 Bug Fixes

- fix #6057 unlink-link order for python noarch packages on windows (#6277)
- fix #6509 custom_channels incorrect in 'conda config --show' (#6510)

7.81 4.3.31 (2017-12-15)

7.81.1 Improvements

- add delete_trash to conda_env create (#6299)

7.81.2 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)

7.81.3 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)

7.82 4.3.30 (2017-10-17)

7.82.1 Improvements

- address #6056 add additional proxy variables to 'conda info --all' (#6083)

7.82.2 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)

7.83 4.3.29 (2017-10-09)

7.83.1 Bug Fixes

- fix #6096 coerce to millisecond timestamps (#6131)

7.84 4.3.28 (2017-10-06)

7.84.1 Bug Fixes

- fix #5854 remove imports of pkg_resources (#5991)
- fix millisecond timestamps (#6001)

7.85 4.3.27 (2017-09-18)

7.85.1 Bug Fixes

- fix #5980 always delete_prefix_from_linked_data in rm_rf (#5982)

7.86 4.3.26 (2017-09-15)

7.86.1 Deprecations/Breaking Changes

- resolve #5922 prioritize channels within multi-channels (#5923)
- add https://repo.continuum.io/pkgs/main to defaults multi-channel (#5931)

7.86.2 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)

7.86.3 Bug Fixes

- always remove linked prefix entry with rm_rf (#5846)
- resolve #5920 bump repodata pickle version (#5921)
- fix msys2 activate and deactivate (#5950)

7.87 4.3.25 (2017-08-16)

7.87.1 Deprecations/Breaking Changes

- resolve #5834 change default value of 'allow_softlinks' from True to False (#5839)

7.87.2 Improvements

- add non-admin check to optionally disable non-privileged operation (#5724)
- add extra warning message to always_softlink configuration option (#5826)

7.87.3 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)

7.88 4.3.24 (2017-07-31)

7.88.1 Bug Fixes

- fix #5708 package priority sort order (#5733)

7.89 4.3.23 (2017-07-21)

7.89.1 Improvements

- resolve #5391 PackageNotFound and NoPackagesFoundError clean up (#5506)

7.89.2 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)

7.90 4.3.22 (2017-06-12)

7.90.1 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)

7.90.2 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)

7.91 4.3.21 (2017-05-25)

7.91.1 Bug Fixes

- fix #5420 conda-env update error (#5421)
- fix #5425 is admin on win int not callable (#5426)

7.92 4.3.20 (2017-05-23)

7.92.1 Improvements

- resolve #5217 skip user confirm in python_api, force always_yes (#5404)

7.92.2 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)

7.92.3 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)

7.93 4.3.19 (2017-05-18)

7.93.1 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)

7.93.2 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)

7.93.3 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)

7.94 4.3.18 (2017-05-09)

7.94.1 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)

7.94.2 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)

7.94.3 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)

7.95 4.3.17 (2017-04-24)

7.95.1 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)

7.95.2 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)

7.95.3 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)

7.96 4.3.16 (2017-03-30)

7.96.1 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)

7.96.2 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)

7.97 4.3.15 (2017-03-20)

7.97.1 Improvements

- allow pkgs_dirs to be configured using `conda config` (#4895)

7.97.2 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)

7.98 4.3.14 (2017-03-03)

7.98.1 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)

7.98.2 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)

7.98.3 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)

7.99 4.3.13 (2017-02-17)

7.99.1 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)

7.99.2 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)

7.99.3 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)

7.100 4.3.12 (2017-02-14)

7.100.1 Improvements

- prepare conda for uploading to pypi (#4619)
- better general http error message (#4627)
- disable old python noarch warning (#4576)

7.100.2 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)

7.100.3 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)

7.101 4.3.11 (2017-02-09)

7.101.1 Bug Fixes

- fix attribute error in add_defaults_to_specs (#4577)

7.102 4.3.10 (2017-02-07)

7.102.1 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)

7.102.2 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)

7.102.3 Non-User-Facing Changes

- separate integration tests with a marker (#4532)

7.103 4.3.9 (2017-01-31)

7.103.1 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)

7.103.2 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)

7.103.3 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)

7.104 4.3.8 (2017-01-23)

7.104.1 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)

7.105 4.3.7 (2017-01-20)

7.105.1 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)

7.106 4.3.6 (2017-01-18)

7.106.1 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)

7.107 4.3.5 (2017-01-17)

7.107.1 Improvements

- add exception message for corrupt repodata (#4315)

7.107.2 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)

7.107.3 Non-User-Facing Changes

- additional package pinning tests (#4317)

7.108 4.3.4 (2017-01-13)

7.108.1 Improvements

- vendor url parsing from urllib3 (#4289)

7.108.2 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)

7.108.3 Non-User-Facing Changes

- make arch in IndexRecord a StringField instead of EnumField
- improve conda-build compatibility (#4266)

7.109 4.3.3 (2017-01-10)

7.109.1 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)

7.109.2 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)

7.110 4.3.2 (2017-01-06)

7.110.1 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)

7.110.2 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)

7.110.3 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)

7.110.4 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)

7.111 4.3.1 (2016-12-19)

7.111.1 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)

7.111.2 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)

7.112 4.3.0 (2016-12-14) Safety

7.112.1 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)

7.112.2 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.

7.112.3 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)

7.112.4 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)

7.112.5 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)

7.113 4.2.17 (unreleased)

7.113.1 Improvements

- silence pre-link warning for old noarch 4.2.x backport (#4453)

7.113.2 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)

7.113.3 Non-User-Facing Changes

- build 4.2.x against conda-build 2.1.2 and enforce passing (#4462)

7.114 4.2.16 (2017-01-20)

7.114.1 Improvements

- vendor url parsing from urllib3 (#4289)
- workaround for symlink race conditions on activate (#4346)

7.114.2 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)

7.114.3 Non-User-Facing Changes

- start using doctests in test runs and coverage (#4304)
- additional package pinning tests (#4312)

7.115 4.2.15 (2017-01-10)

7.115.1 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)

7.115.2 Bug Fixes

- replace sed with python for activate and deactivate #4257

7.116 4.2.14 (2017-01-07)

7.116.1 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)

7.116.2 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)

7.116.3 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)

7.117 4.2.13 (2016-11-22)

7.117.1 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)

7.117.2 Improvements

- double/extend http timeouts (#3831)
- let descriptive http errors cover more http exceptions (#3834)
- backport some conda-build configuration (#3875)

7.117.3 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)

7.117.4 Non-User-Facing Changes

- flake8 E116, E121, & E123 enabled (#3883)

7.118 4.2.12 (2016-11-02)

7.118.1 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)

7.119 4.2.11 (2016-10-23)

7.119.1 Improvements

- only try once for windows trash removal (#3698)

7.119.2 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)

7.120 4.2.10 (2016-10-18)

7.120.1 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)

7.120.2 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)

7.120.3 Non-User-Facing Changes

- backport conda.exports module to 4.2.x (#3654)
- travis-ci OSX fix (#3615 via #3657)

7.121 4.2.9 (2016-09-27)

7.121.1 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)

7.122 4.2.8 (2016-09-26)

7.122.1 Improvements

- add "error" key back to json error output (#3523)

7.122.2 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)

7.123 4.2.7 (2016-09-16)

7.123.1 Deprecations/Breaking Changes

- revert to 4.1.x behavior of `conda list --export` (#3450, #3451)

7.123.2 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)

7.123.3 Non-User-Facing Changes

- use install.rm_rf for TemporaryDirectory cleanup (#3425)

7.124 4.2.6 (2016-09-14)

7.124.1 Improvements

- add support for client TLS certificates (#3419)
- address #3267 allow migration of channel_alias (#3410)
- conda-env version matches conda version (#3422)

7.124.2 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)

7.125 4.2.5 (2016-09-08)

7.125.1 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)

7.125.2 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)

7.125.3 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)

7.126 4.2.4 (2016-08-18)

7.126.1 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)

7.126.2 Non-User-Facing Changes

- additional conda-env integration tests (#3288)

7.127 4.2.3 (2016-08-11)

7.127.1 Improvements

- added zsh and zsh.exe to Windows shells (#3257)

7.127.2 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)

7.127.3 Non-User-Facing Changes

- add integration tests for conda-env (#3270)
- add more conda-build smoke tests (#3274)

7.128 4.2.2 (2016-08-09)

7.128.1 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)

7.128.2 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)

7.129 4.2.1 (2016-08-01)

7.129.1 Improvements

- improve an error message that can happen during conda install --revision (#3181)
- use clean sys.exit with user choice 'No' (#3196)

7.129.2 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)

7.129.3 Non-User-Facing Changes

- move scripts in bin to shell directory (#3186)

7.130 4.2.0 (2016-07-28) Configuration

7.130.1 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)

7.130.2 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)

7.130.3 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)

7.130.4 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)

7.130.5 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)

7.131 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

7.132 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

7.133 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

7.134 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

7.135 4.1.9 (2016-07-20)

- fix #3104, add global BINSTAR_TOKEN_PAT
- handle failures in binstar_client more generally

7.136 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

7.137 4.0.11 2016-07-09

- allow auto_update_conda from sysrc, #3015 via #3021

7.138 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

7.139 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

7.140 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

7.141 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

7.142 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

7.143 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

7.144 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

7.145 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

7.146 4.0.9 (2016-06-15)

- add auto_update_conda config parameter, #2686

7.147 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

7.148 4.0.8 (2016-06-03)

- fix a potential problem with moving files to trash, #2587

7.149 4.0.7 (2016-05-26)

- workaround for boto bug, #2380

7.150 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

7.151 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

7.152 4.0.4 (2016-03-10)

- revert #2217 closing request sessions, #2233

7.153 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

7.154 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

7.155 4.0.1 (2016-03-07)

- fixes an install issue caused by md5 checksum mismatches, #2183
- remove auxlib build dependency, #2188

7.156 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

7.157 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

7.158 3.19.3 (2016-02-19)

- fix critical issue, see #2106

7.159 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

7.160 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

7.161 3.19.0 (2015-12-17)

- OpenBSD 5.x support, #1891
- improve install CLI to make Miniconda -f work, #1905

7.162 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

7.163 3.18.8 (2015-12-03)

- always attempt to delete files in rm_rf, #1864

7.164 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

7.165 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

7.166 3.18.5 (2015-11-11)

- fix bad menuinst exception handling, #1798
- add workaround for unresolved dependencies on Windows

7.167 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

7.168 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

7.169 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

7.170 3.18.1 (2015-09-28)

- fix: removed reference to win_ignore_root in plan module

7.171 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.

7.172 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

7.173 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)

7.174 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

7.175 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

7.176 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

7.177 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

7.178 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%

7.179 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.

7.180 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

7.181 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

7.182 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

7.183 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

7.184 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

7.185 3.8.4 (2015-02-03)

- copy (de)activate scripts from conda-env
- Add noarch (sub) directory support

7.186 3.8.3 (2015-01-28)

- simplified how ROOT_PREFIX is obtained in (de)activate

7.187 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

7.188 3.8.1 (2015-01-23)

- add missing utf-8 decoding, fixes Python 3 bug when icondata to json file

7.189 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

7.190 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

7.191 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

7.192 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

7.193 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.

7.194 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

7.195 3.6.4 (2014-09-08)

- fix removing packages that aren't in the channels any more
- Pretties output for --alt-hint

7.196 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

7.197 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

7.198 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

7.199 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

7.200 3.5.5 (2014-06-10)

- remove another instance pycosat version detection, which fails on Windows, see issue #761

7.201 3.5.4 (2014-06-10)

- remove pycosat version detection, which fails on Windows, see issue #761

7.202 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

7.203 3.5.2 (2014-05-27)

- fix bug in activate.bat and deactivate.bat on Windows

7.204 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

7.205 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

7.206 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

7.207 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

7.208 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

7.209 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

7.210 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

7.211 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.

7.212 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.

7.213 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

7.214 3.2.0 (2014-03-11)

- print dots to the screen for progress
- move logic functions from resolve to logic module

7.215 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.

7.216 3.1.1 (2014-03-07)

- check if directory exists, fixed issue #591

7.217 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

7.218 3.0.6 (2014-02-20)

- fix 'conda update' taking build number into account

7.219 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

7.220 3.0.4 (2014-02-14)

- conda update: don't try to update packages that are already up-to-date

7.221 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.

7.222 3.0.2 (2014-02-03)

- fix detection of prefix being writable

7.223 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

7.224 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

7.225 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.

7.226 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

7.227 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

7.228 2.2.7 (2014-01-02)

- fixed bug in conda build related to lchmod not being available on all platforms

7.229 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

7.230 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

7.231 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

7.232 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

7.233 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

7.234 2.2.1 (2013-11-15)

- minor fixes related to make conda pip installable
- generated conda meta-data missing ‘files’ key, fixed issue #357

7.235 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

7.236 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.

7.237 2.0.4 (2013-11-04)

- ensure lowercase name when generating package info, fixed issue #329
- on Windows, handle the .nonadmin files

7.238 2.0.3 (2013-10-28)

- update bundle format
- fix bug when displaying packages to be downloaded (thanks Crystal)

7.239 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

7.240 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)

7.241 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

7.242 1.9.1 (2013-09-06)

- fix bug in new caching of repodata index

7.243 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

7.244 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

7.245 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

7.246 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.

7.247 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

7.248 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

7.249 1.7.0 (2013-06-20)

- allow ${PREFIX} in app_entry
- add binstar upload information after conda build finishes

7.250 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

7.251 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)

7.252 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)

7.253 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)

7.254 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

7.255 1.5.2 (2013-04-29)

- fixed issue 59: bad error message when pkgs dir is not writable

7.256 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)

7.257 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'

7.258 1.4.6 (2013-03-12)

- fixed channel selection (issue #56)

7.259 1.4.5 (2013-03-11)

- fix issue #53 with install for meta packages
- add -q/–quiet option to update command

7.260 1.4.4 (2013-03-09)

- use numpy 1.7 as default on all platforms

7.261 1.4.3 (2013-03-09)

- fixed bug in conda.builder.share.clone_bundle()

7.262 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

7.263 1.4.1 (2013-03-07)

- fix some feature selection bugs
- never exit in activate and deactivate
- improve help and error messages

7.264 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

7.265 1.3.5 (2013-02-05)

- fixed detecting untracked files on Windows
- removed backwards compatibility to conda 1.0 version

7.266 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)

7.267 1.3.3 (2013-01-23)

- fix conda create not filtering channels correctly
- remove (hidden) --test and --testgui options

7.268 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

7.269 1.3.1 (2013-01-22)

- fix bug in conda update not installing new dependencies

7.270 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

7.271 1.2.1 (2012-11-21)

- remove ambiguity from conda update command

7.272 1.2.0 (2012-11-20)

- “conda upgrade” now updates from AnacondaCE to Anaconda (removed upgrade2pro
- add versioneer

7.273 1.1.0 (2012-11-13)

- Many new features implemented by Bryan

7.274 1.0.0 (2012-09-06)

- initial release

PYTHON MODULE INDEX

C

INDEX

O

P

Q

R

S

U

V