conda Documentation

Release 4.8.2.post14+e9a50561

Continuum Analytics

Jan 30, 2020

CONTENTS

1 User guide 3

2 Conda configuration 113

3 Conda Python API 123

4 Release notes 131

5 Command reference 239

6 Glossary 257

Python Module Index 261

Index 263

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 `-n` are the same, and `--envs` and `-e` are the same.

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

- *What is a conda package?*
- *.conda file format*
- *Using packages*
- *Package structure*
- *Metapackages*
  - *Anaconda metapackage*
  - *Mutex metapackages*
- *Noarch packages*
  - *Noarch Python*
- *Link and unlink scripts*

- *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*
- *Repository structure and index*
- *Package match specifications*
- *Version ordering*

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=3.4`. Internally, conda translates the command line syntax to the spec defined in this section.

EXAMPLE: python=3.4 is translated to python 3.4*.

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"
```

(continues on next page)

(continued from previous page)

```
conda install "numpy=1.11.1|1.11.3"
conda install "numpy>=1.8,<2"
```

Examples

The OR constraint "numpy=1.11.1|1.11.3" matches with 1.11.1 or 1.11.3.

The AND constraint "numpy>=1.8,<2" matches with 1.8 and 1.9 but not 2.0.

The fuzzy constraint numpy=1.11 matches 1.11, 1.11.0, 1.11.1, 1.11.2, 1.11.18, and so on.

The exact constraint numpy==1.11 matches 1.11, 1.11.0, 1.11.0.0, and so on.

The build string constraint "numpy=1.11.2=*nomkl*" matches the NumPy 1.11.2 packages without MKL but not the normal MKL NumPy 1.11.2 packages.

The build string constraint "numpy=1.11.1|1.11.3=py36_0" matches NumPy 1.11.1 or 1.11.3 built for Python 3.6 but not any versions of NumPy built for Python 3.5 or Python 2.7.

The following are all valid match specifications for numpy-1.8.1-py27_0:

- numpy
- numpy 1.8*
- numpy 1.8.1
- numpy >=1.8
- numpy ==1.8.1
- numpy 1.8|1.8*
- numpy >=1.8,<2
- numpy >=1.8,<2|1.9
- numpy 1.8.1 py27_0
- numpy=1.8.1=py27_0

Version ordering

The `class VersionOrder(object)` implements an order relation between version strings.

Version strings can contain the usual alphanumeric characters (A-Za-z0-9), separated into components by dots and underscores. Empty segments (i.e. two consecutive dots, a leading/trailing underscore) are not permitted. An optional epoch number - an integer followed by `!` - can precede the actual version string (this is useful to indicate a change in the versioning scheme itself). Version comparison is case-insensitive.

Supported version strings

Conda supports six types of version strings:

- Release versions contain only integers, e.g. `1.0`, `2.3.5`.
- Pre-release versions use additional letters such as `a` or `rc`, for example `1.0a1`, `1.2.beta3`, `2.3.5rc3`.
- Development versions are indicated by the string `dev`, for example `1.0dev42`, `2.3.5.dev12`.
- Post-release versions are indicated by the string `post`, for example `1.0post1`, `2.3.5.post2`.
- Tagged versions have a suffix that specifies a particular property of interest, e.g. `1.1.parallel`. Tags can be added to any of the preceding 4 types. As far as sorting is concerned, tags are treated like strings in pre-release versions.
- An optional local version string separated by + can be appended to the main (upstream) version string. It is only considered in comparisons when the main versions are equal, but otherwise handled in exactly the same manner.

Predictable version ordering

To obtain a predictable version ordering, it is crucial to keep the version number scheme of a given package consistent over time. Conda considers prerelease versions as less than release versions.

- Version strings should always have the same number of components (except for an optional tag suffix or local version string).
- Letters/Strings indicating non-release versions should always occur at the same position.

Before comparison, version strings are parsed as follows:

- They are first split into epoch, version number, and local version number at `!` and + respectively. If there is no `!`, the epoch is set to 0. If there is no +, the local version is empty.
- The version part is then split into components at `.` and _.
- Each component is split again into runs of numerals and non-numerals
- Subcomponents containing only numerals are converted to integers.
- Strings are converted to lowercase, with special treatment for `dev` and `post`.
- When a component starts with a letter, the fillvalue 0 is inserted to keep numbers and strings in phase, resulting in `1.1.a1' == 1.1.0a1'`.
- The same is repeated for the local version part.

Examples:

```
1.2g.beta15.rc => [[0], [1], [2, 'g'], [0, 'beta', 15], [0, 'rc']]
1!2.15.1_ALPHA => [[1], [2], [15], [1, '_alpha']]
```

The resulting lists are compared lexicographically, where the following rules are applied to each pair of corresponding subcomponents:

- Integers are compared numerically.
- Strings are compared lexicographically, case-insensitive.
- Strings are smaller than integers, except
  - `dev` versions are smaller than all corresponding versions of other types.
  - `post` versions are greater than all corresponding versions of other types.

- If a subcomponent has no correspondent, the missing correspondent is treated as integer 0 to ensure `'1.1' == 1.1.0'`.

The resulting order is:

```
  0.4
< 0.4.0
< 0.4.1.rc
== 0.4.1.RC   # case-insensitive comparison
< 0.4.1
< 0.5a1
< 0.5b3
< 0.5C1      # case-insensitive comparison
< 0.5
< 0.9.6
< 0.960923
< 1.0
< 1.1dev1    # special case ``dev``
< 1.1a1
< 1.1.0dev1  # special case ``dev``
== 1.1.dev1   # 0 is inserted before string
< 1.1.a1
< 1.1.0rc1
< 1.1.0
== 1.1
< 1.1.0post1 # special case ``post``
== 1.1.post1  # 0 is inserted before string
< 1.1post1   # special case ``post``
< 1996.07.12
< 1!0.4.1    # epoch increased
< 1!3.1.1.6
< 2!0.4.1    # epoch increased again
```

Some packages (most notably OpenSSL) have incompatible version conventions. In particular, OpenSSL interprets letters as version counters rather than pre-release identifiers. For OpenSSL, the relation `1.0.1 < 1.0.1a => True # for OpenSSL` holds, whereas conda packages use the opposite ordering. You can work around this problem by appending a dash to plain version numbers:

```
1.0.1a => 1.0.1post.a # ensure correct ordering for OpenSSL
```

1.1.4 Conda channels

- *What is a "conda channel"?*
- *Specifying channels when installing packages*
- *Conda clone channel RSS feed*

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 default set of channels, and packages are automatically downloaded and updated from https://repo.anaconda.com/pkgs/. You can modify what remote channels are automatically searched. You might want to do this to maintain 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>
```

(continues on next page)

(continued from previous page)

```
    </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>
```

1.1.5 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?

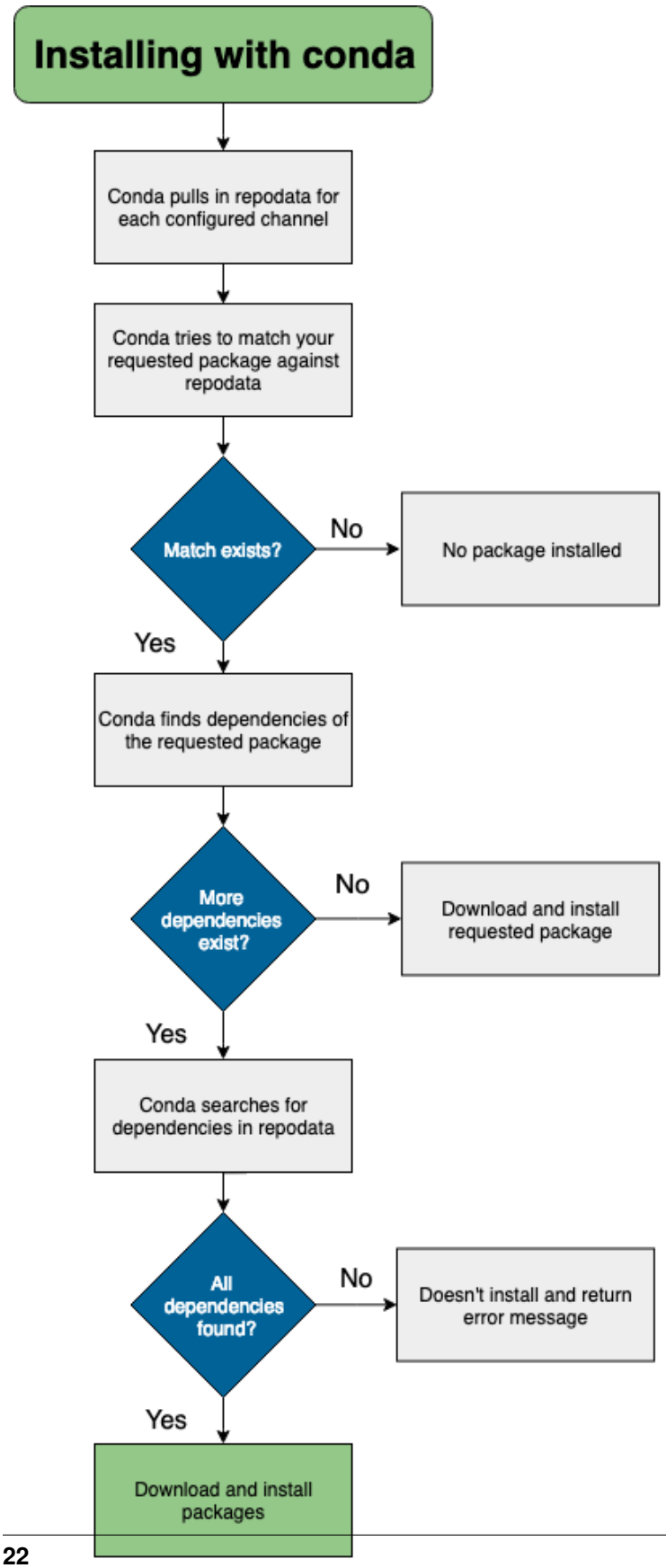
Package system differentiators

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.

1.1.6 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 3.5.2.
- If Python 3.4.0 is currently installed, and the latest version of Python is 3.5.2, then `conda install python=3` installs Python 3.5.2.

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
```

1.1.7 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.

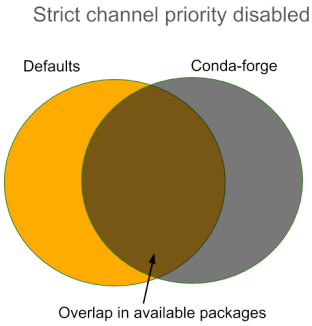
Are you:

- **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.

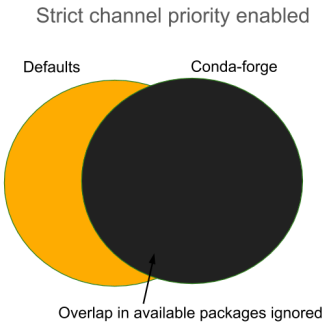
Read more about how we made conda faster.

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.

1.1.8 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.

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.

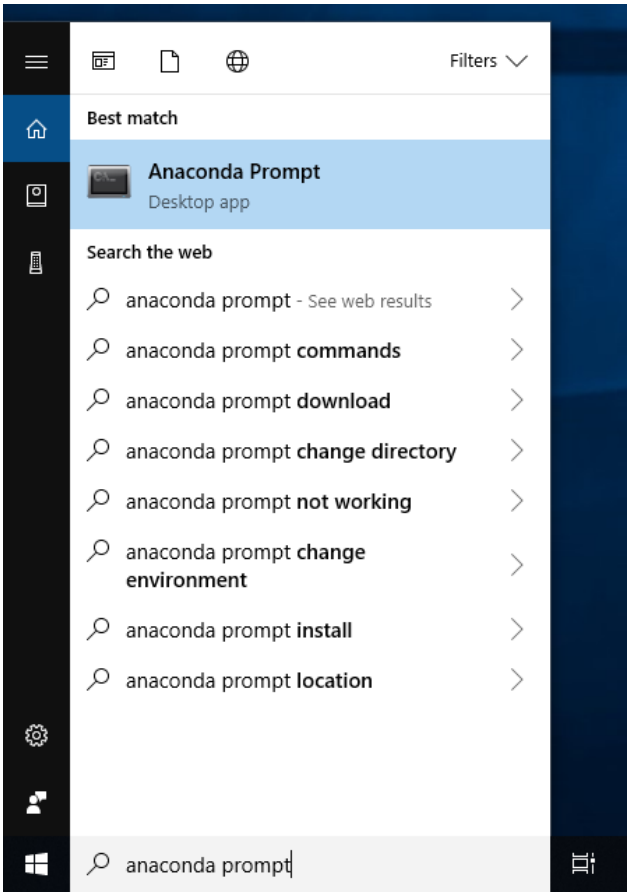
MacOS

- Open Launchpad, then click the terminal icon.

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

Linux

- Open a terminal window.


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

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

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 3.5:

```
conda create --name snakes python=3.5
```

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

- *System requirements*
- *Regular installation*
- *Installing in silent mode*
- *Installing conda on a system that has other Python installations or packages*

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.

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

- *Anaconda or Miniconda?*
- *Choosing a version of Anaconda or Miniconda*
- *GUI versus command line installer*
- *Choosing a version of Python*
- *Cryptographic hash verification*

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, which includes Python 3.7. 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.

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 *Using conda on Windows XP with or without a 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 3.7, which is included with Anaconda3 and Miniconda3.
- You can easily set up additional versions of Python such as 3.5 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.

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 `1`
- `/RegisterPython=[0|1]`---Make this the system's default Python. `0` indicates `JustMe`, which is the default. `1` indicates `AllUsers`.
- `/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 `~/.bashrc`. Assumes that you agree to the license agreement. Does not edit the `.bashrc` or `.bash_profile` files.
- -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

If you are on macOS Catalina, the new default shell is zsh. You will instead need to run `source <path to conda>/bin/activate` followed by `conda init zsh`.

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-get 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
```

(continues on next page)

(continued from previous page)

```
[conda]
name=Conda
baseurl=https://repo.anaconda.com/pkgs/misc/rpmrepo/conda
enabled=1
gpgcheck=1
gpgkey=https://repo.anaconda.com/pkgs/misc/gpgkeys/anaconda.asc
EOF
```

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

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

  Dependencies Resolved


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

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

  Transaction Summary

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

Check to see if the installation is successful by typing:

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

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

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

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

These RPM and Debian packages 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*
- *Creating and editing*
- *Searching for .condarc*
- *General configuration*
- *Advanced configuration*
- *Conda-build configuration*
- *Expansion of environment variables*
- *Obtaining information from the .condarc file*
- *Configuring number of threads*

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',
     '/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/',
     '~/.conda/.condarc',
     '~/.conda/condarc',
```

(continues on next page)

(continued from previous page)

```
    '~/.conda/condarc.d/',
    '~/.condarc',
    '$CONDA_PREFIX/.condarc',
    '$CONDA_PREFIX/condarc',
    '$CONDA_PREFIX/condarc.d/',
    '$CONDARC',
)
```

`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 superceded 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)*
- *Allow other channels (allow_other_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
```

Allow other channels (allow_other_channels)

The system-level `.condarc` file may specify a set of allowed channels, and it may allow users to install packages from other channels with the boolean flag `allow_other_channels`. The default is `True`.

If `allow_other_channels` is set to `False`, only those channels explicitly specified in the system `.condarc` file are allowed:

```
allow_other_channels: False
```

When `allow_other_channels` is set to `True` or not specified, each user has access to the default channels and to any channels that the user specifies in their local `.condarc` file. When `allow_other_channels` is set to `false`, if the user specifies other channels, the other channels are blocked and the user receives a message reporting that channels are blocked. For more information, see *Example administrator-controlled installation*.

If the system `.condarc` file specifies a `channel_alias`, it overrides any channel aliases set in a user's `.condarc` file. See *Set a channel alias (channel_alias)*.

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)*
- *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
```

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:

- `envs_dirs`
- `pkgs_dirs`
- `ssl_verify`
- `client_cert`
- `client_cert_key`
- `proxy_servers`
- `channels`
- `custom_channels`
- `custom_multichannels`
- `default_channels`
- `migrated_custom_channels`
- `whitelist_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

- *Adding the free channel to defaults*
- *Changing .condarc*
- *Package name changes*
- *Troubleshooting*

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 1 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 sets `allow_other_channels` to `False`, and it specifies that users may download packages only from the `admin` channel:

```
$ cat /tmp/miniconda/.condarc
allow_other_channels : false
channel_alias: https://conda.anaconda.org/
channels:
  - admin
```

   Note: The admin channel can also be expressed as https://conda.anaconda.org/admin/

   Because `allow_other_channels` is `False` 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 is deprecated starting with version 4.4.

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 Using conda on Windows XP with or without a proxy

Although Windows XP mainstream support and Extended Support from Microsoft have ended, and Windows XP is no longer one of the target systems supported by Anaconda, some users have had success using Anaconda on Windows XP with the methods described on this page.

Anaconda 2.3.0 is the last version of Python 3-based Anaconda to support Windows XP. Anaconda 2.4 and later have a version of Python 3 built with Visual Studio 2015, which by default does not support Windows XP.

You can install Anaconda 2.3.0 and then update it with `conda update conda` and `conda update --all`. Download `Anaconda3-2.3.0-Windows-x86.exe` at https://repo.anaconda.com/archive/. Install it in any location, such as `C:\Anaconda`.

Using a proxy with Windows XP

To configure conda for use behind a corporate proxy that uses proxy auto-config (PAC) files and SSL certificates for secure connections:

1. Find a proxy server address from the PAC file:
    1. Open Internet Explorer.
    2. From the **Tools** menu, select Internet Options, and then click the **Connections** tab.
    3. On the **Connections** tab, click the LAN Settings button.
    4. In the LAN Settings dialog box, copy the address under the Use automatic configuration script checkbox.
    5. Click the Cancel button to close the LAN settings.
    6. Click the Cancel button to close the Internet Options.
    7. Paste the address into the Internet Explorer address bar, then press the Enter key.
    8. In the PAC file that opens, search for `return` until you find what looks like a proxy IP or DNS address with the port number, which may take some time in a long file.
    9. Copy the address and port number.
2. Follow the *.condarc instructions* to create a file named `.condarc` in your home directory or the installation directory, such as `C:\Anaconda\.condarc`.
3. Follow the *.condarc proxy server instructions* to add proxy information to the `.condarc` file.

If you decide to include any passwords, be aware of transformations that can affect special characters.

EXAMPLE: This example shows proxy information with passwords:

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

ssl_verify: False
```

If you include proxy information without passwords, you will be asked to answer authentication prompts at the command line.

EXAMPLE: This example shows proxy information without passwords:

```
proxy_servers:
  http: http://corp.com:8080
  https: https://corp.com:8080

ssl_verify: False
```

Once the proxy is configured, you can run `conda update conda` and then create and manage environments with the Anaconda Navigator.

Some packages such as `flask-login` may not be available through conda, so you may need to use pip to install them:

1. To use pip securely over https:

    ```
    pip install --trusted-host pypi.python.org package-name
    ```

2. If the secure https proxy fails, you can force pip to use an insecure http proxy instead:

```
pip install --index-url=http://pypi.python.org/simple/ --trusted-host pypi.python.
↪org package-name
```

1.4.8 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.

1.4.9 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` 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 (.bshrs or e.g. .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*
- *Determining your conda version*
- *Updating conda to the current version*
- *Suppressing warning message about updating 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 4.7.12`.

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

- *Creating an environment with commands*
- *Creating an environment from an environment.yml file*
- *Specifying a location for an environment*
- *Updating an environment*
- *Cloning an environment*
- *Building identical conda environments*
- *Activating an environment*
- *Deactivating an environment*

- *Determining your current environment*
- *Viewing a list of your environments*
- *Viewing a list of the packages in an environment*
- *Using pip in an environment*
- *Setting environment variables*
- *Saving environment variables*
- *Sharing an environment*
- *Restoring an environment*
- *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/`. This environment uses the same version of Python that you are currently using because you did not specify a version.

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

```
conda create -n myenv python=3.6
```

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=0.15.0
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy=0.15.0
```

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

```
conda create -n myenv python=3.6 scipy=0.15.0 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=0.35 matplotlib=3.1 numpy=1.16
```

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 reactivate 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` 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.

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=3.6   # or 2.7
  - bokeh=0.9.2
  - numpy=1.9.*
  - nodejs=0.10.*
  - flask
```

(continues on next page)

(continued from previous page)

```
  - pip:
    - Flask-Testing
```

Note: Note the use of the wildcard * when defining the patch version number. Defining the version number by fixing the major and minor version numbers while allowing the patch version number to vary allows us to use our environment file to update our environment to get any bug fixes whilst still maintaining consistency of software environment.

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 https://repo.anaconda.com/pkgs/. Read more about *conda channels*.

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 it as a default. However, it may break old environment files, so we plan to delay making it conda's out-of-the-box default until the next major version bump, conda 5.0.

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

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

channel_priority: flexible
```

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
   ```

(continues on next page)

(continued from previous page)

```
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

- *Searching for packages*
- *Installing packages*
- *Installing similar packages*
- *Installing packages from Anaconda.org*
- *Installing non-conda packages*
- *Installing commercial packages*
- *Viewing a list of installed packages*
- *Listing package dependencies*

- *Updating packages*
- *Preventing packages from updating (pinning)*
- *Adding default packages to new environments automatically*
- *Removing 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 info package_name`
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 2.7 updates to the highest available in the 2.x series and 3.6 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

- *Viewing a list of available Python versions*
- *Installing a different version of Python*
- *Using a different version of Python*
- *Updating or upgrading Python*

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

Anaconda supports Python 2.7, 3.6, and 3.7. The default is Python 2.7 or 3.7, depending on which installer you used:

- For the installers "Anaconda" and "Miniconda," the default is 2.7.
- For the installers "Anaconda3" or "Miniconda3," the default is 3.7.

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 3.6, in your terminal window or an Anaconda Prompt, run:

```
conda create -n py36 python=3.6 anaconda
```

> **Note:** Replace `py36` with the name of the environment you want to create. `anaconda` is the meta-package that includes all of the Python packages comprising the Anaconda distribution. `python=3.6` is the package and version you want to install in this new environment. This could be any package, such as `numpy=1.7`, or *multiple packages*.

   - To create the new environment for Python 2.7, in your terminal window or an Anaconda Prompt, run:

```
conda create -n py27 python=2.7 anaconda
```

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
```

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---3.6---by installing that version of Python:

```
conda install python=3.6
```

1.5.7 Managing virtual packages

- *Listing detected virtual packages*
- *Overriding detected 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.

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

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
```

(continues on next page)

(continued from previous page)

```
            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

- *The .travis.yml file*
- *Supporting packages that do not have official builds*
- *Building a conda recipe*
- *AppVeyor*
- *Bootstrap your environment*

If you are already using Travis CI, using conda is a preferable alternative to using apt-get 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 2.7, 3.5, and 3.6:

```
language: python
python:
  # We don't actually use the Travis Python, but this keeps it organized.
  - "2.7"
  - "3.5"
  - "3.6"
install:
  - sudo apt-get update
  # We do this conditionally because it saves us some downloading if the
  # version is the same.
  - if [[ "$TRAVIS_PYTHON_VERSION" == "2.7" ]]; then
      wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh -O␣
↪miniconda.sh;
```

(continues on next page)

(continued from previous page)

```
    else
      wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh -O␣
↪miniconda.sh;
    fi
  - 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 setup.py 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
```

AppVeyor

AppVeyor is a continuous build service for Windows built on Azure and is an alternative to using Travis CI with conda.

For an example project building conda packages on AppVeyor, see https://github.com/rmcgibbo/python-appveyor-conda-example.

Bootstrap your environment

To bootstrap your environment, use the standalone conda approach in your `appveyor.yml`:

```
# Config file for automatic testing at travis-ci.org

language: python
python:
  - "2.7"
  - "3.7"

install:
  - wget https://repo.anaconda.com/pkgs/misc/conda-execs/conda-latest-linux-64.exe -O␣
↪conda.exe
  - chmod +x conda.exe
  - export CONDA_ALWAYS_YES=1
  # This is where you put any extra dependencies you may have.
  - ./conda.exe create -p $HOME/miniconda python=$TRAVIS_PYTHON_VERSION conda conda-
↪build pytest six pytest-cov pytest-mock
  - export PATH="$HOME/miniconda/bin:$PATH"
  - hash -r
  # Install your code here.
script:
  - pytest -v --color=yes --cov=cpr tests
after_success:
  - conda install codecov
  - codecov
```

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 cheat sheet` PDF (1 MB) for a single-page summary of the most important information about using conda.

1.7 Troubleshooting

- *Using conda in Windows Batch script exits early*
- *NumPy MKL library load failed*
- *SSL connection errors*
- *Permission denied errors during installation*
- *Permission denied errors after using sudo conda command*
- *Already installed error message*
- *Conda reports that a package is installed, but it appears not to be*
- *pkg_resources.DistributionNotFound: conda==3.6.1-6-gb31b0d4-dirty*
- *macOS error "ValueError unknown locale: UTF-8"*
- *AttributeError or missing getproxies*
- *Shell commands open from the wrong location*
- *Programs fail due to invoking conda Python instead of system Python*
- *UnsatisfiableSpecifications error*
- *Package installation fails from a specific channel*
- *Conda automatically upgrades to unwanted version*
- *Conda upgrade error*
- *ValidationError: Invalid value for timestamp*
- *Unicode error after installing Python 2*
- *Windows environment has not been activated*
- *The system cannot find the path specified on Windows*

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.

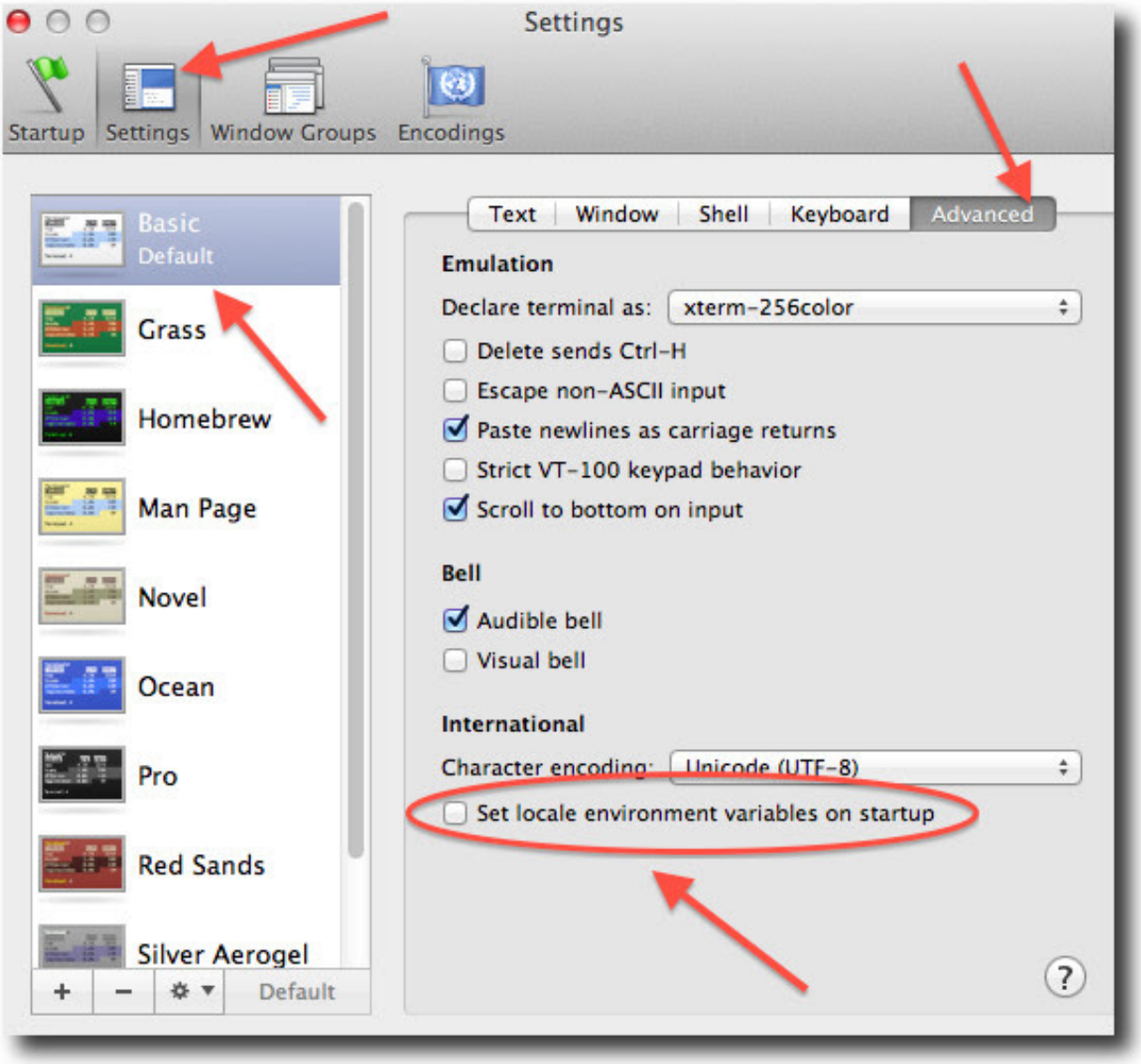
This sets your `LANG` environment variable to be empty. This may cause Terminal to use incorrect settings for your locale. The `locale` command in Terminal tells you what settings are used.

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.


Settings
Startup
Settings
Window Groups
Encodings
Basic
Default
Grass
Homebrew
Man Page
Novel
Ocean
Pro
Red Sands
Silver Aerogel
Default
Text
Window
Shell
Keyboard
Advanced
Emulation
Declare terminal as:
xterm-256color
Delete sends Ctrl-H
Escape non-ASCII input
Paste newlines as carriage returns
Strict VT-100 keypad behavior
Scroll to bottom on input
Bell
Audible bell
Visual bell
International
Character encoding:
Unicode (UTF-8)
Set locale environment variables on startup

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 info <package>" 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 info wxpython` or `conda info wxpython=3` 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:

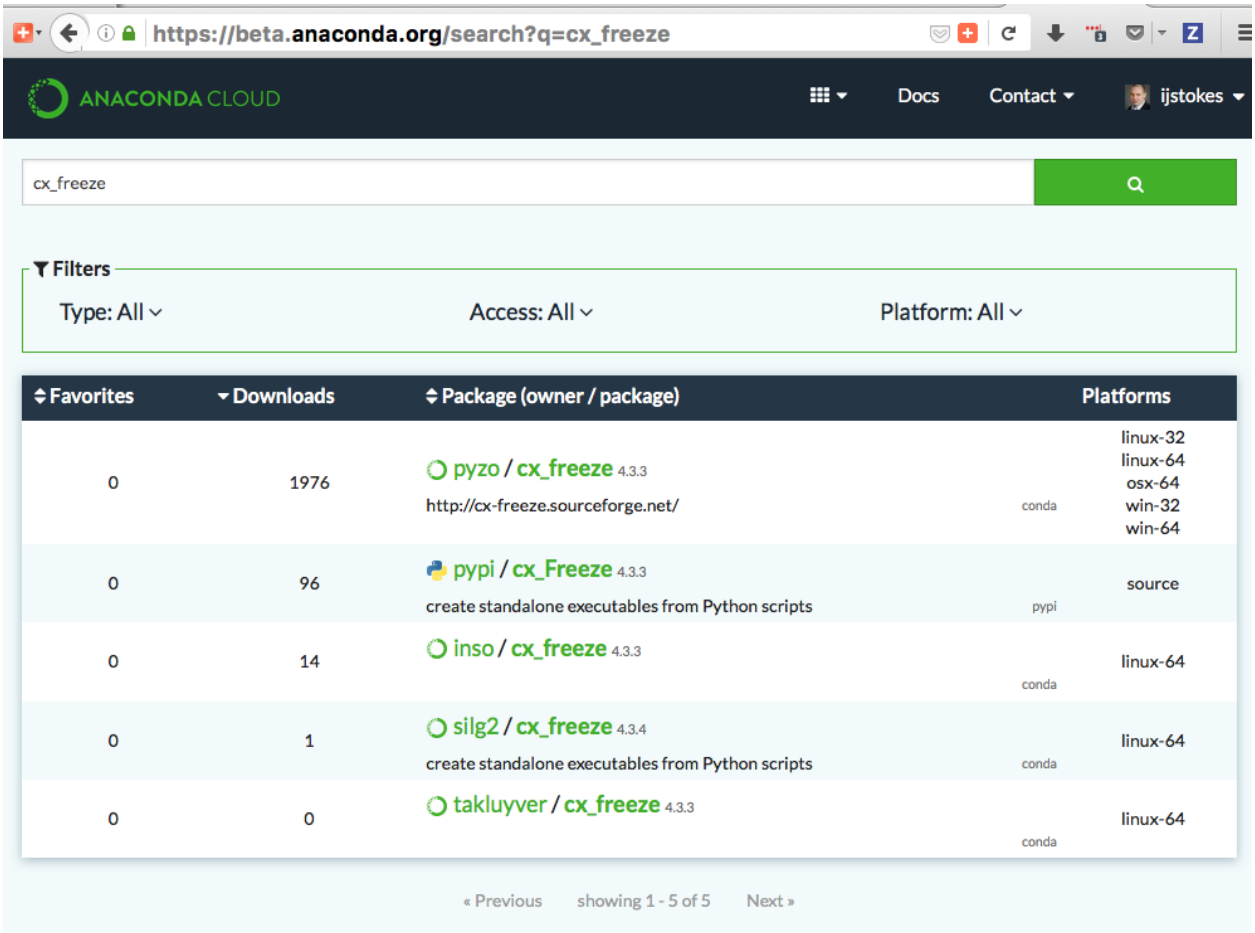
```
$ conda create -c pyzo -n cxfreeze_py34 cx_freeze python=3.4
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata: ..........
Solving package specifications: .........

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
```

(continues on next page)


https://beta.anaconda.org/search?q=cx_freeze
ANACONDA CLOUD
Docs
Contact
ijstokes
cx_freeze
Filters
Type: All
Access: All
Platform: All
Favorites
Downloads
Package (owner / package)
Platforms
0
1976
pyzo / cx_freeze 4.3.3
http://cx-freeze.sourceforge.net/
conda
linux-32
linux-64
osx-64
win-32
win-64
0
96
pypi / cx_Freeze 4.3.3
create standalone executables from Python scripts
pypi
source
0
14
inso / cx_freeze 4.3.3
conda
linux-64
0
1
silg2 / cx_freeze 4.3.4
create standalone executables from Python scripts
conda
linux-64
0
0
takluyver / cx_freeze 4.3.3
conda
linux-64
« Previous
showing 1 - 5 of 5
Next »

(continued from previous page)

```
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

# # whitelist_channels (sequence: primitive)
# #   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.
# #
# whitelist_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
```

(continues on next page)

(continued from previous page)

```
# #   '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.
# #
# 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
```

(continues on next page)

(continued from previous page)

```
# # 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


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

# # envs_dirs (sequence: primitive)
# #   aliases: envs_path
# #   env var string delimiter: ':'
# #   The list of directories to search for named environments. When
# #   creating a new named environment, the environment will be placed in
# #   the first writable location.
# #
# envs_dirs: []

# # pkgs_dirs (sequence: primitive)
# #   env var string delimiter: ','
# #   The list of directories where locally-available packages are linked
# #   from at install time. Packages not locally available are downloaded
# #   and extracted into the first writable directory.
# #
# pkgs_dirs: []

# # default_threads (int)
# #   Threads to use by default for parallel operations.  Default is None,
# #   which allows operations to choose themselves.  For more specific
# #   control, see the other *_threads parameters:     * repodata_threads -
# #   for fetching/loading repodata     * verify_threads - for verifying
# #   package contents in transactions     * execute_threads - for carrying
# #   out the unlinking and linking steps
# #
# default_threads: 0
```

(continues on next page)

(continued from previous page)

```
# ######################################################
# ##               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 conjuction with
# #   client_ssl_cert for individual files.
# #
# client_ssl_cert:

# # client_ssl_cert_key (NoneType, str)
# #   aliases: client_cert_key
# #   Used in conjunction with client_ssl_cert for a matching key file.
# #
# client_ssl_cert_key:

# # local_repodata_ttl (bool, int)
# #   For a value of False or 0, always fetch remote repodata (HTTP 304
# #   responses respected). For a value of True or 1, respect the HTTP
# #   Cache-Control max-age header. Any other positive integer values is the
# #   number of seconds to locally cache repodata before checking the remote
# #   server for an update.
# #
# local_repodata_ttl: 1

# # offline (bool)
# #   Restrict conda to cached download content and file:// based urls.
# #
# offline: false

# # proxy_servers (map: primitive)
# #   A mapping to enable proxy settings. Keys can be either (1) a
# #   scheme://hostname form, which will match any request to the given
# #   scheme and exact hostname, or (2) just a scheme, which will match
# #   requests to that scheme. Values are are the actual proxy server, and
# #   are of the form 'scheme://[user:password@]host[:port]'. The optional
# #   'user:password' inclusion enables HTTP Basic Auth with your proxy.
# #
# proxy_servers: {}

# # remote_connect_timeout_secs (float)
# #   The number seconds conda will wait for your client to establish a
# #   connection to a remote url resource.
# #
# remote_connect_timeout_secs: 9.15

# # remote_max_retries (int)
# #   The maximum number of retries each HTTP connection should attempt.
# #
# remote_max_retries: 3

# # remote_backoff_factor (int)
# #   The factor determines the time HTTP connection should wait for
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
# # create_default_packages (sequence: primitive)
# #   env var string delimiter: ','
# #   Packages that are by default added to a newly created environments.
# #
# create_default_packages: []

# # disallowed_packages (sequence: primitive)
# #   aliases: disallow
# #   env var string delimiter: '&'
# #   Package specifications to disallow installing. The default is to allow
# #   all packages.
# #
# disallowed_packages: []

# # force_reinstall (bool)
# #   Ensure that any user-requested package for the current operation is
# #   uninstalled and reinstalled, even if that package already exists in
# #   the environment.
# #
# force_reinstall: false

# # pinned_packages (sequence: primitive)
# #   env var string delimiter: '&'
# #   A list of package specs to pin for every environment resolution. This
# #   parameter is in BETA, and its behavior may change in a future release.
# #
# pinned_packages: []

# # 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: []


# ######################################################
# ##  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).
# #
```

(continues on next page)

(continued from previous page)

```
# allow_softlinks: false

# # always_copy (bool)
# #   aliases: copy
# #   Register a preference that files be copied into a prefix during
# #   install rather than hard-linked.
# #
# always_copy: false

# # always_softlink (bool)
# #   aliases: softlink
# #   Register a preference that files be soft-linked (symlinked) into a
# #   prefix during install rather than hard-linked. The link source is the
# #   'pkgs_dir' package cache from where the package is being linked.
# #   WARNING: Using this option can result in corruption of long-lived
# #   conda environments. Package caches are *caches*, which means there is
# #   some churn and invalidation. With this option, the contents of
# #   environments can be switched out (or erased) via operations on other
# #   environments.
# #
# always_softlink: false

# # path_conflict (PathConflict)
# #   The method by which conda handle's conflicting/overlapping paths
# #   during a create, install, or update operation. The value must be one
# #   of 'clobber', 'warn', or 'prevent'. The '--clobber' command-line flag
# #   or clobber configuration parameter overrides path_conflict set to
# #   'prevent'.
# #
# path_conflict: clobber

# # rollback_enabled (bool)
# #   Should any error occur during an unlink/link transaction, revert any
# #   disk mutations made to that point in the transaction.
# #
# 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

# # 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
```

(continues on next page)

(continued from previous page)

```
# #   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
# #   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: {}
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
# #
# 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
# #   higher this number, the longer the generation of the unsat hint will
# #   take. Defaults to 3.
# #
# unsatisfiable_hints_check_depth: 2
```

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 conda.core.solve.**DepsModifier**

Flags to enable alternate handling of dependencies.

NOT_SET = 'not_set'

NO_DEPS = 'no_deps'

ONLY_DEPS = 'only_deps'

class conda.core.solve.**Solver**(*prefix*, *channels*, *subdirs=()*, *specs_to_add=()*, *specs_to_remove=()*, *repodata_fn='repodata.json'*, *command=<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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<auxlib._Null object>*, *force_reinstall=<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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<auxlib._Null object>*, *force_reinstall=<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 conda.cli.python_api.**Commands**

CLEAN = 'clean'

CONFIG = 'config'

CREATE = 'create'

HELP = 'help'

INFO = 'info'

INSTALL = 'install'

LIST = 'list'

REMOVE = 'remove'

RUN = 'run'

SEARCH = 'search'

UPDATE = 'update'

conda.cli.python_api.**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 comamnd on the command line see below for examples. Be very careful to delimit arguements 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 occured, 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 conda.api.**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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<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=<auxlib._Null object>*, *deps_modifier=<auxlib._Null object>*, *prune=<auxlib._Null object>*, *ignore_pinned=<auxlib._Null object>*, *force_remove=<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 conda.api.**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 conda.api.**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=<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 conda.api.**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=<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

RELEASE NOTES

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

4.1 4.8.2 (2020-01-24)

4.1.1 Enhancements

- Improved solver messaging (#9560)

4.1.2 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)
- Added code of conduct (#9601, #9602, #9603, #9603, #9604, #9605)

4.1.3 Bug fixes

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

4.1.4 Contributors

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

4.2 4.8.1 (2019-12-19)

4.2.1 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)

4.2.2 Bug fixes

- Fix calling Python API run_command with list and string arguments (#9331)
- Set tmp to shortened path that excludes spaces (#9409)
- Add subdir to PackageRecord dist_str (#9418)
- Revert init bash completion (#9421)
- Avoid function redefinition upon resourcing conda.fish (#9444)
- Propagate PIP error level when creating envs with conda env (#9460)
- Fix incorrect chown call (#9464)
- Don't check in pkgs for trash (#9472)
- Fix running conda activate in multiple processes on Windows (#9477)
- 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)

4.2.3 Docs

- Fix string concatenation running words together regarding CONDA_EXE (#9411)
- Fix typo ("list" -> "info") (#9433)
- Improve description of DLL loading verification and activating environments (#9453)
- Installing with specific build number (#9534)
- Typo in condarc key envs_dirs (#9478)
- Clarify channel priority and package sorting (#9492)

4.2.4 Contributors

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

4.3 4.8.0 (2019-11-04)

4.3.1 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)

4.3.2 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)
- Fix xonsh activation and tab completion (#9305)
- Fix what channels are queried when context.offline is True (#9385)

4.3.3 Docs

- Removed 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)

4.3.4 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

4.4 4.7.12 (2019-09-12)

4.4.1 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)

4.4.2 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)

4.4.3 Contributors

- @beenje
- @Bezier89
- @cjmartian
- @forrestwaters
- @jjhelmus
- @martin-raden
- @msarahan
- @nganani

- @rrigdon
- @soapy1
- @WesRoach
- @zheaton

4.5 4.7.11 (2019-08-06)

4.5.1 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)

4.5.2 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)

4.5.3 Docs

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

4.5.4 Contributors

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

4.6 4.7.10 (2019-07-19)

4.6.1 Bug fixes

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

4.6.2 Contributors

- @msarahan

4.7 4.7.9 (2019-07-18)

4.7.1 Bug fixes

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

4.7.2 Contributors

- @jjhelmus
- @msarahan

4.8 4.7.8 (2019-07-17)

4.8.1 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)

4.8.2 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)

4.8.3 Contributors

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

4.9 4.7.7 (2019-07-12)

4.9.1 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)

4.9.2 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)

4.9.3 Contributors

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

4.10 4.7.6 (2019-07-11)

4.10.1 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)

4.10.2 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)

4.10.3 Contributors

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

4.11 4.7.5 (2019-06-24)

4.11.1 Improvements

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

4.11.2 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)

4.11.3 Contributors

- @dhirschfeld
- @msarahan
- @rrigdon

4.12 4.7.4 (2019-06-19)

4.12.1 Improvements

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

4.12.2 Bug fixes

- Fix tcsh/csh init scripts (#8792)

4.12.3 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.

4.12.4 Contributors

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

4.13 4.7.3 (2019-06-14)

4.13.1 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)

4.13.2 Contributors

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

4.14 4.7.2 (2019-06-10)

4.14.1 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)

4.14.2 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)

4.14.3 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; 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)

4.14.4 Contributors

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

4.15 4.7.1 (2019-05-30)

4.15.1 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)

4.15.2 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)

4.15.3 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.

4.15.4 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)

4.15.5 Contributors

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

4.16 4.7.0 (2019-05-17)

4.16.1 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)

4.16.2 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)
- Fix *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)

4.16.3 Contributors

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

4.17 4.6.14 (2019-04-17)

4.17.1 Bug fixes

- Export extra function in powershell Conda.psm1 script (fixes Anaconda powershell prompt) (#8570)

4.17.2 Contributors

- @msarahan

4.18 4.6.13 (2019-04-16)

4.18.1 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)

4.18.2 Contributors

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

4.19 4.6.12 (2019-04-10)

4.19.1 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)

4.19.2 Docs improvements

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

4.19.3 Contributors

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

4.20 4.6.11 (2019-04-04)

4.20.1 Bug fixes

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

4.20.2 Docs improvements

- Changelogs for 4.6.10

4.20.3 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

4.21 4.6.10 (2019-04-01)

4.21.1 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 condabinconda.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)

4.21.2 Docs improvements

- Changelogs for 4.6.8, 4.6.9

4.21.3 Contributors

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

4.22 4.6.9 (2019-03-29)

4.22.1 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)

4.22.2 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)

4.22.3 Docs improvements

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

4.22.4 Contributors

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

4.23 4.6.8 (2019-03-06)

4.23.1 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)

4.23.2 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)

4.23.3 Contributors

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

4.24 4.6.7 (2019-02-21)

4.24.1 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)

4.24.2 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)

4.24.3 Contributors

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

4.25 4.6.6 (2019-02-18)

4.25.1 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)

4.25.2 Contributors

- @mingwandroid
- @msarahan

4.26 4.6.5 (2019-02-15)

4.26.1 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)

4.26.2 Contributors

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

4.27 4.6.4 (2019-02-13)

4.27.1 Improvements

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

4.27.2 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)

4.27.3 Deprecations/Breaking changes

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

4.27.4 Contributors

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

4.28 4.6.3 (2019-02-07)

4.28.1 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)

4.28.2 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)

4.28.3 Contributors

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

4.29 4.6.2 (2019-01-29)

4.29.1 Improvements

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

4.29.2 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)

4.29.3 Contributors

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

4.30 4.5.13 (2019-01-29)

4.30.1 Improvements

- Document the allow_conda_downgrades configuration parameter (#8034)
- Remove conda upgrade message (#8161)

4.30.2 Contributors

- @msarahan
- @nehaljwani

4.31 4.6.1 (2019-01-21)

4.31.1 Improvements

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

4.31.2 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)

4.31.3 Contributors

- @jjhelmus
- @msarahan
- @pv

4.32 4.6.0 (2019-01-15)

4.32.1 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)

4.32.2 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)

4.32.3 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)

4.32.4 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)

4.32.5 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)

4.32.6 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

4.32.7 Contributors

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

4.33 4.5.12 (2018-12-10)

4.33.1 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)

4.33.2 Bug fixes

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

4.33.3 Contributors

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

4.34 4.5.11 (2018-08-21)

4.34.1 Improvements

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

4.34.2 Contributors

- @CJ-Wright
- @mbargull

4.35 4.5.10 (2018-08-13)

4.35.1 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)

4.35.2 Contributors

- @jesse-
- @nehaljwani

4.36 4.5.9 (2018-07-30)

4.36.1 Improvements

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

4.36.2 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)

4.36.3 Contributors

- @kalefranz
- @loriab

4.37 4.5.8 (2018-07-10)

4.37.1 Bug fixes

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

4.37.2 Contributors

- @kalefranz

4.38 4.5.7 (2018-07-09)

4.38.1 Improvements

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

4.38.2 Bug fixes

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

4.38.3 Contributors

- @kalefranz
- @nehaljwani

4.39 4.5.6 (2018-07-06)

4.39.1 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)

4.39.2 Contributors

- @kalefranz

4.40 4.5.5 (2018-06-29)

4.40.1 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)

4.40.2 Contributors

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

4.41 4.5.4 (2018-05-14)

4.41.1 Improvements

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

4.41.2 Bug fixes

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

4.41.3 Contributors

- @jakirkham
- @kalefranz
- @mbargull

4.42 4.5.3 (2018-05-07)

4.42.1 Bug fixes

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

4.43 4.5.2 (2018-04-27)

4.43.1 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)

4.44 4.5.1 (2018-04-13)

4.44.1 Improvements

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

4.44.2 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)

4.45 4.5.0 (2018-03-20)

4.45.1 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.

4.45.2 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)

4.45.3 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)

4.45.4 API

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

4.45.5 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)

4.45.6 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)

4.46 4.4.11 (2018-02-23)

4.46.1 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)

4.46.2 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)

4.47 4.4.10 (2018-02-09)

4.47.1 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)

4.48 4.4.9 (2018-02-06)

4.48.1 Improvements

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

4.48.2 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)

4.49 4.4.8 (2018-01-25)

4.49.1 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)

4.49.2 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)

4.50 4.4.7 (2018-01-08)

4.50.1 Improvements

- Resolve #6650 add upgrade message for unicode errors in Python 2 (#6651)

4.50.2 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 '<unknown>' (#6668, #6673)
- Fix #6627 AttributeError on 'conda activate' (#6655)

4.51 4.4.6 (2017-12-31)

4.51.1 Bug fixes

- Fix #6612 do not assume Anaconda Python on Windows nor Librarybin hack (#6615)
- Recipe test improvements and associated bug fixes (#6614)

4.52 4.4.5 (2017-12-29)

4.52.1 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)

4.53 4.4.4 (2017-12-24)

4.53.1 Improvements

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

4.53.2 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)

4.54 4.4.3 (2017-12-22)

4.54.1 Improvements

- Adjust error report message (#6534)

4.54.2 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)

4.54.3 Non-user-facing changes

- Regression tests for #6512 (#6515)

4.55 4.4.2 (2017-12-22)

4.55.1 Deprecations/Breaking changes

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

4.55.2 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)

4.56 4.4.1 (2017-12-21)

4.56.1 Bug fixes

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

4.57 4.4.0 (2017-12-20)

4.57.1 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.

4.57.2 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). 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.

4.57.3 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)

4.57.4 API

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

4.57.5 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)

4.57.6 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)

4.57.7 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)

4.58 4.3.34 (2018-02-09)

4.58.1 Bug fixes

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

4.59 4.3.33 (2018-01-24)

4.59.1 Bug fixes

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

4.60 4.3.32 (2018-01-10)

4.60.1 Improvements

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

4.60.2 Bug fixes

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

4.61 4.3.31 (2017-12-15)

4.61.1 Improvements

- Add delete_trash to conda_env create (#6299)

4.61.2 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)

4.61.3 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)

4.62 4.3.30 (2017-10-17)

4.62.1 Improvements

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

4.62.2 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)

4.63 4.3.29 (2017-10-09)

4.63.1 Bug fixes

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

4.64 4.3.28 (2017-10-06)

4.64.1 Bug fixes

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

4.65 4.3.27 (2017-09-18)

4.65.1 Bug fixes

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

4.66 4.3.26 (2017-09-15)

4.66.1 Deprecations/Breaking changes

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

4.66.2 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)

4.66.3 Bug fixes

- Always remove linked prefix entry with rm_rf (#5846)
- Resolve #5920 bump repodata pickle version (#5921)
- Fix msys2 activate and deactivate (#5950)

4.67 4.3.25 (2017-08-16)

4.67.1 Deprecations/Breaking changes

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

4.67.2 Improvements

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

4.67.3 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)

4.68 4.3.24 (2017-07-31)

4.68.1 Bug fixes

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

4.69 4.3.23 (2017-07-21)

4.69.1 Improvements

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

4.69.2 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)

4.70 4.3.22 (2017-06-12)

4.70.1 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)

4.70.2 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)

4.71 4.3.21 (2017-05-25)

4.71.1 Bug fixes

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

4.72 4.3.20 (2017-05-23)

4.72.1 Improvements

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

4.72.2 Bug fixes

- Fix #5367 conda info always shows 'unknown' for admin indicator on Windows (#5368)
- Fix #5248 drop plan description information that might not alwasy 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)

4.72.3 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)

4.73 4.3.19 (2017-05-18)

4.73.1 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)

4.73.2 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)

4.73.3 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)

4.74 4.3.18 (2017-05-09)

4.74.1 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)

4.74.2 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)

4.74.3 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)

4.75 4.3.17 (2017-04-24)

4.75.1 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)

4.75.2 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)

4.75.3 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)

4.76 4.3.16 (2017-03-30)

4.76.1 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)

4.76.2 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)

4.77 4.3.15 (2017-03-20)

4.77.1 Improvements

- Allow pkgs_dirs to be configured using *conda config* (#4895)

4.77.2 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)

4.78 4.3.14 (2017-03-03)

4.78.1 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)

4.78.2 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)

4.78.3 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)

4.79 4.3.13 (2017-02-17)

4.79.1 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)

4.79.2 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)

4.79.3 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)

4.80 4.3.12 (2017-02-14)

4.80.1 Improvements

- Prepare conda for uploading to PyPI (#4619)
- Better general http error message (#4627)
- Disable old Python noarch warning (#4576)

4.80.2 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)

4.80.3 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)

4.81 4.3.11 (2017-02-09)

4.81.1 Bug fixes

- Fix attribute error in add_defaults_to_specs (#4577)

4.82 4.3.10 (2017-02-07)

4.82.1 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)

4.82.2 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)

4.82.3 Non-user-facing changes

- Separate integration tests with a marker (#4532)

4.83 4.3.9 (2017-01-31)

4.83.1 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)

4.83.2 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)

4.83.3 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)

4.84 4.3.8 (2017-01-23)

4.84.1 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)

4.85 4.3.7 (2017-01-20)

4.85.1 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)

4.86 4.3.6 (2017-01-18)

4.86.1 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)

4.87 4.3.5 (2017-01-17)

4.87.1 Improvements

- Add exception message for corrupt repodata (#4315)

4.87.2 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)

4.87.3 Non-user-facing changes

- Additional package pinning tests (#4317)

4.88 4.3.4 (2017-01-13)

4.88.1 Improvements

- Vendor url parsing from urllib3 (#4289)

4.88.2 Bug fixes

- Fix some bugs in windows multi-user support (#4277)
- Fix problems with channels of type <unknown> (#4290)
- Include aliases for first command-line argument (#4279)
- Fix for multi-line FTP status codes (#4276)

4.88.3 Non-user-facing changes

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

4.89 4.3.3 (2017-01-10)

4.89.1 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)

4.89.2 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)

4.90 4.3.2 (2017-01-06)

4.90.1 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)

4.90.2 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)

4.90.3 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)

4.90.4 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)

4.91 4.3.1 (2016-12-19)

4.91.1 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)

4.91.2 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)

4.92 4.3.0 (2016-12-14) Safety

4.92.1 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)

4.92.2 Deprecations/Breaking changes

- Conda now has the ability to refuse to clobber existing files that are not within the unlink instructions of the transaction. This behavior is configurable via the *path_conflict* configuration option, which has three possible values: *clobber*, *warn*, and *prevent*. In 4.3, the default value will be *clobber*. That will give package maintainers time to correct current incompatibilities within their package ecosystem. In 4.4, the default will switch to *warn*, which means these operations continue to clobber, but the warning messages are displayed. In *4.5*, the default value will switch to *prevent*. As we tighten up the *path_conflict* constraint, a new command line flag *--clobber* will loosen it back up on an *ad hoc* basis. Using *--clobber* overrides the setting for *path_conflict* to effectively be *clobber* for that operation.
- 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.

4.92.3 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)

4.92.4 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)

4.92.5 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)

4.93 4.2.15 (2017-01-10)

4.93.1 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)

4.93.2 Bug fixes

- Replace sed with Python for activate and deactivate #4257

4.94 4.2.14 (2017-01-07)

4.94.1 Improvements

- Use install.rm_rf for TemporaryDirectory cleanup (#3425)
- Improve handling of local dependency information (#2107)
- Add default channels to exports for Windows Linux and macOS (#4103)
- Make subdir configurable (#4178)

4.94.2 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)

4.94.3 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)

4.95 4.2.13 (2016-11-22)

4.95.1 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)

4.95.2 Improvements

- Double/extend http timeouts (#3831)
- Let descriptive http errors cover more http exceptions (#3834)
- Backport some conda-build configuration (#3875)

4.95.3 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)

4.95.4 Non-user-facing changes

- Flake8 E116, E121, & E123 enabled (#3883)

4.96 4.2.12 (2016-11-02)

4.96.1 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)

4.97 4.2.11 (2016-10-23)

4.97.1 Improvements

- Only try once for Windows trash removal (#3698)

4.97.2 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)

4.98 4.2.10 (2016-10-18)

4.98.1 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)

4.98.2 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)

4.98.3 Non-user-facing changes

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

4.99 4.2.9 (2016-09-27)

4.99.1 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)

4.100 4.2.8 (2016-09-26)

4.100.1 Improvements

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

4.100.2 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)

4.101 4.2.7 (2016-09-16)

4.101.1 Deprecations/Breaking changes

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

4.101.2 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)

4.101.3 Non-user-facing changes

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

4.102 4.2.6 (2016-09-14)

4.102.1 Improvements

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

4.102.2 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)

4.103 4.2.5 (2016-09-08)

4.103.1 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)

4.103.2 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)

4.103.3 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)

4.104 4.2.4 (2016-08-18)

4.104.1 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)

4.104.2 Non-user-facing changes

- Additional conda-env integration tests (#3288)

4.105 4.2.3 (2016-08-11)

4.105.1 Improvements

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

4.105.2 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)

4.105.3 Non-user-facing changes

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

4.106 4.2.2 (2016-08-09)

4.106.1 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)

4.106.2 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)

4.107 4.2.1 (2016-08-01)

4.107.1 Improvements

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

4.107.2 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)

4.107.3 Non-user-facing changes

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

4.108 4.2.0 (2016-07-28)

4.108.1 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)

4.108.2 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)

4.108.3 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 Linux and macOS (#3076)

4.108.4 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)

4.108.5 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)

4.109 4.1.12 (2016-09-08)

- Fix #2837 "File exists" in symlinked path with parallel activations, #3210
- Fix prune option when installing packages, #3354
- Change check for placeholder to be more friendly to long PATH, #3349

4.110 4.1.11 (2016-07-26)

- Fix PS1 backup in activate script, #3135 via #3155
- Correct resolution for 'handle failures in binstar_client more generally', #3156

4.111 4.1.10 (2016-07-25)

- Ignore symlink failure because of read-only file system, #3055
- Backport shortcut tests, #3064
- Fix #2979 redefinition of $SHELL variable, #3081
- Fix #3060 --clone root --copy exception, #3080

4.112 4.1.9 (2016-07-20)

- Fix #3104, add global BINSTAR_TOKEN_PAT
- Handle failures in binstar_client more generally

4.113 4.1.8 (2016-07-12)

- Fix #3004 UNAUTHORIZED for url (null binstar token), #3008
- Fix overwrite existing redirect shortcuts when symlinking envs, #3025
- Partially revert no default shortcuts, #3032, #3047

4.114 4.1.7 (2016-07-07)

- Add msys2 channel to defaults on Windows, #2999
- Fix #2939 channel_alias issues; improve offline enforcement, #2964
- Fix #2970, #2974 improve handling of file:// URLs inside channel, #2976

4.115 4.1.6 (2016-07-01)

- Slow down exp backoff from 1 ms to 100 ms factor, #2944
- Set max time on exp_backoff to ~6.5 sec,#2955
- Fix #2914 add/subtract from PATH; kill folder output text, #2917
- Normalize use of get_index behavior across clone/explicit, #2937
- Wrap root prefix check with normcase, #2938

4.116 4.1.5 (2016-06-29)

- More conservative auto updates of conda #2900
- Fix some permissions errors with more aggressive use of move_path_to_trash, #2882
- Fix #2891 error if allow_other_channels setting is used, #2896
- Fix #2886, #2907 installing a tarball directly from the package cache, #2908
- Fix #2681, #2778 reverting #2320 lock behavior changes, #2915

4.117 4.1.4 (2016-06-27)

- Fix #2846 revert the use of UNC paths; shorten trash filenames, #2859
- Fix exp backoff on Windows, #2860
- Fix #2845 URL for local file repos, #2862
- Fix #2764 restore full path var on win; create to CONDA_PREFIX env var, #2848
- Fix #2754 improve listing pip installed packages, #2873
- Change root prefix detection to avoid clobbering root activate scripts, #2880
- Address #2841 add lowest and highest priority indication to channel config output, #2875
- Add SYMLINK_CONDA to planned instructions, #2861
- Use CONDA_PREFIX, not CONDA_DEFAULT_ENV for activate.d, #2856
- Call scripts with redirect on win; more error checking to activate, #2852

4.118 4.1.3 (2016-06-23)

- Ensure conda-env auto update, along with conda, #2772
- Make yaml booleans behave how everyone expects them to, #2784
- Use accept-encoding for repodata; prefer repodata.json to repodata.json.bz2, #2821
- Additional integration and regression tests, #2757, #2774, #2787
- Add offline mode to printed info; use offline flag when grabbing channels, #2813
- Show conda-env version in conda info, #2819
- Adjust channel priority superseded list, #2820
- Support epoch ! characters in command line specs, #2832
- Accept old default names and new ones when canonicalizing channel URLs #2839
- Push PATH, PS1 manipulation into shell scripts, #2796
- Fix #2765 broken source activate without arguments, #2806
- Fix standalone execution of install.py, #2756
- Fix #2810 activating conda environment broken with git bash on Windows, #2795
- Fix #2805, #2781 handle both file-based channels and explicit file-based URLs, #2812

- Fix #2746 conda create --clone of root, #2838
- Fix #2668, #2699 shell recursion with activate #2831

4.119 4.1.2 (2016-06-17)

- Improve messaging for "downgrades" due to channel priority, #2718
- Support conda config channel append/prepend, handle duplicates, #2730
- Remove --shortcuts option to internal CLI code, #2723
- Fix an issue concerning space characters in paths in activate.bat, #2740
- Fix #2732 restore yes/no/on/off for booleans on the command line, #2734
- Fix #2642 tarball install on Windows, #2729
- Fix #2687, #2697 WindowsError when creating environments on Windows, #2717
- Fix #2710 link instruction in conda create causes TypeError, #2715
- Revert #2514, #2695, disabling of .netrc files, #2736
- Revert #2281 printing progress bar to terminal, #2707

4.120 4.1.1 (2016-06-16)

- Add auto_update_conda config parameter, #2686
- Fix #2669 conda config --add channels can leave out defaults, #2670
- Fix #2703 ignore activate symlink error if links already exist, #2705
- Fix #2693 install duplicate packages with older version of Anaconda, #2701
- Fix #2677 respect HTTP_PROXY, #2695
- Fix #2680 broken fish integration, #2685, #2694
- Fix an issue with conda never exiting, #2689
- Fix #2688 explicit file installs, #2708
- Fix #2700 conda list UnicodeDecodeError, #2706

4.121 4.1.0 (2016-06-14)

This release contains many small bug fixes for all operating systems, and a few special fixes for Windows behavior.

4.121.1 Notable changes for all systems (Windows, macOS, and Linux)

- **Channel order now matters.** The most significant conda change is that when you add channels, channel order matters. If you have a list of channels in a .condarc file, conda installs the package from the first channel where it's available, even if it's available in a later channel with a higher version number.
- **No version downgrades.** Conda remove no longer performs version downgrades on any remaining packages that might be suggested to resolve dependency losses; the package will just be removed instead.
- **New YAML parser/emitter.** PyYAML is replaced with ruamel.yaml, which gives more robust control over yaml document use. More on ruamel.yaml
- **Shebang lines over 127 characters are now truncated (Linux, macOS only).** Shebangs are the first line of the many executable scripts that tell the operating system how to execute the program. They start with `#!`. Most OSes don't support these lines over 127 characters, so conda now checks the length and replaces the full interpreter path in long lines with `/usr/bin/env`. When you're working in a conda environment that is deeply under many directories, or you otherwise have long paths to your conda environment, make sure you activate that environment now.
- **Changes to conda list command.** When looking for packages that aren't installed with conda, conda list now examines the Python site-packages directory rather than relying on pip.
- **Changes to conda remove command.** The command `conda remove --all` now removes a conda environment without fetching information from a remote server on the packages in the environment.
- **Conda update can be turned off and on.** When turned off, conda will not update itself unless the user manually issues a conda update command. Previously conda updated any time a user updated or installed a package in the root environment. Use the option `conda config set auto_update_conda false`.
- **Improved support for BeeGFS.** BeeGFS is a parallel cluster file system for performance and designed for easy installation and management. More on BeeGFS

4.121.2 Windows-only changes

- **Shortcuts are no longer installed by default on Windows.** Shortcuts can now be installed with the `--shortcuts` option. Example 1: Install a shortcut to Spyder with `conda install spyder --shortcut`. Note if you have Anaconda (not Miniconda), you already have this shortcut and Spyder. Example 2: Install the open source package named `console_shortcut`. When you click the shortcut icon, a terminal window will open with the environment containing the `console_shortcut` package already activated. `conda install console_shortcut --shortcuts`
- **Skip binary replacement on Windows.** Linux & macOS have binaries that are coded with library locations, and this information must sometimes be replaced for relocatability, but Windows does not generally embed prefixes in binaries, and was already relocatable. We skip binary replacement on Windows.

Complete list:

- Clean up activate and deactivate scripts, moving back to conda repo, #1727, #2265, #2291, #2473, #2501, #2484
- Replace pyyaml with ruamel_yaml, #2283, #2321
- Better handling of channel collisions, #2323, #2369 #2402, #2428
- Improve listing of pip packages with conda list, #2275
- Re-license progressbar under BSD 3-clause, #2334
- Reduce the amount of extraneous info in hints, #2261
- Add --shortcuts option to install shortcuts on windows, #2623

- Skip binary replacement on windows, #2630
- Don't show channel urls by default in conda list, #2282
- Package resolution and solver tweaks, #2443, #2475, #2480
- Improved version & build matching, #2442, #2488
- Print progress to the terminal rather than stdout, #2281
- Verify version specs given on command line are valid, #2246
- Fix for try_write function in case of odd permissions, #2301
- Fix a conda search --spec error, #2343
- Update User-Agent for conda connections, #2347
- Remove some dead code paths, #2338, #2374
- Fixes a thread safety issue with http requests, #2377, #2383
- Manage BeeGFS hard-links non-POSIX configuration, #2355
- Prevent version downgrades during removes, #2394
- Fix conda info --json, #2445
- Truncate shebangs over 127 characters using /usr/bin/env, #2479
- Extract packages to a temporary directory then rename, #2425, #2483
- Fix help in install, #2460
- Fix re-install bug when sha1 differs, #2507
- Fix a bug with file deletion, #2499
- Disable .netrc files, #2514
- Dont fetch index on remove --all, #2553
- Allow track_features to be a string *or* a list in .condarc, #2541
- Fix #2415 infinite recursion in invalid_chains, #2566
- Allow channel_alias to be different than binstar, #2564

4.122 4.0.11 (2016-07-09)

- Allow auto_update_conda from sysrc, #3015 via #3021

4.123 4.0.10 (2016-06-29)

- Fix #2846 revert the use of UNC paths; shorten trash filenames, #2859 via #2878
- Fix some permissions errors with more aggressive use of move_path_to_trash, #2882 via #2894

4.124 4.0.9 (2016-06-15)

- Add auto_update_conda config parameter, #2686

4.125 4.0.8 (2016-06-03)

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

4.126 4.0.7 (2016-05-26)

- Workaround for boto bug, #2380

4.127 4.0.6 (2016-05-11)

- Log "custom" versions as updates rather than downgrades, #2290
- Fixes a TypeError exception that can occur on install/update, #2331
- Fixes an error on Windows removing files with long path names, #2452

4.128 4.0.5 (2016-03-16)

- Improved help documentation for install, update, and remove, #2262
- Fixes #2229 and #2250 related to conda update errors on Windows, #2251
- Fixes #2258 conda list for pip packages on Windows, #2264

4.129 4.0.4 (2016-03-10)

- Revert #2217 closing request sessions, #2233

4.130 4.0.3 (2016-03-10)

- Adds a *conda clean --all* feature, #2211
- Solver performance improvements, #2209
- Fixes conda list for pip packages on windows, #2216
- Quiets some logging for package downloads under Python 3, #2217
- More urls for *conda list --explicit*, #1855
- Prefer more "latest builds" for more packages, #2227
- Fixes a bug with dependency resolution and features, #2226

4.131 4.0.2 (2016-03-08)

- Fixes track_features in ~/.condarc being a list, see also #2203
- Fixes incorrect path in lock file error #2195
- Fixes issues with cloning environments, #2193, #2194
- Fixes a strange interaction between features and versions, #2206
- Fixes a bug in low-level SAT clause generation creating a preference for older versions, #2199

4.132 4.0.1 (2016-03-07)

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

4.133 4.0.0 (2016-03-04)

- 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 depedencies properly according to the setting of the "update_deps" configuration:

  --update-deps: conda will also update any dependencies as needed to install the latest verison of the requrested packages. The minimal set of changes required to achieve this is sought.

  —no-update-deps: conda will update the packages *only* to the extent that no updates to the dependencies are required

  The previous behavior, which would update the packages without regard to their dependencies, could result in a broken configuration, and has been removed.
- Conda finally has an official logo.
- Fix *conda clean --packages* on Windows, #1944
- Conda sub-commands now support dashes in names, #1840

4.134 3.19.3 (2016-02-19)

- Fix critical issue, see #2106

4.135 3.19.2 (2016-02-19)

- Add basic activate/deactivate, conda activate/deactivate/ls for fish, see #545
- Remove error when CONDA_FORCE_32BIT is set on 32-bit systems, #1985
- Suppress help text for --unknown option, #2051
- Fix issue with conda create --clone post-link scripts, #2007
- Fix a permissions issue on windows, #2083

4.136 3.19.1 (2016-02-01)

- Resolve.py: properly escape periods in version numbers, #1926
- Support for pinning Lua by default, #1934
- Remove hard-coded test URLs, a module cio_test is now expected when CIO_TEST is set

4.137 3.19.0 (2015-12-17)

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

4.138 3.18.9 (2015-12-10)

- Allow chaning default_channels (only applies to "system" condarc), from from CLI, #1886
- Improve default for --show-channel-urls in conda list, #1900

4.139 3.18.8 (2015-12-03)

- Always attempt to delete files in rm_rf, #1864

4.140 3.18.7 (2015-12-02)

- Simplify call to menuinst.install()
- Add menuinst as dependency on Windows
- Add ROOT_PREFIX to post-link (and pre_unlink) environment

4.141 3.18.6 (2015-11-19)

- Improve conda clean when user lacks permissions, #1807
- Make show_channel_urls default to True, #1771
- Cleaner write tests, #1735
- Fix documentation, #1709
- Improve conda clean when directories don't exist, #1808

4.142 3.18.5 (2015-11-11)

- Fix bad menuinst exception handling, #1798
- Add workaround for unresolved dependencies on Windows

4.143 3.18.4 (2015-11-09)

- Allow explicit file to contain MD5 hashsums
- Add --md5 option to "conda list --explicit"
- Stop infinite recursion during certain resolve operations, #1749
- Add dependencies even if strictness == 3, #1766

4.144 3.18.3 (2015-10-15)

- Added a pruning step for more efficient solves, #1702
- Disallow conda-env to be installed into non-root environment
- Improve error output for bad command input, #1706
- Pass env name and setup cmd to menuinst, #1699

4.145 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 <pkg>.tar.bz2", unlink any installed package with that name, not just the installed one
- Allow menu items to be installed in non-root env, #1692

4.146 3.18.1 (2015-09-28)

- Fix: removed reference to win_ignore_root in plan module

4.147 3.18.0 (2015-09-28)

- Allow Python to be updated in root environment on Windows, #1657
- Add defaults to specs after getting pinned specs (allows to pin a different version of Python than what is installed)
- Show what older versions are in the solutions in the resolve debug log
- Fix some issues with Python 3.5
- Respect --no-deps when installing from .tar or .tar.bz2
- Avoid infinite recursion with NoPackagesFound and conda update --all --file
- Fix conda update --file
- Toposort: Added special case to remove 'pip' dependency from 'Python'
- Show dotlog messages during hint generation with --debug
- Disable the max_only heuristic during hint generation
- New version comparison algorithm, which consistently compares any version string, and better handles version strings using things like alpha, beta, rc, post, and dev. This should remove any inconsistent version comparison that would lead to conda installing an incorrect version.
- Use the trash in rm_rf, meaning more things will get the benefit of the trash system on Windows
- Add the ability to pass the --file argument multiple times
- Add conda upgrade alias for conda update
- Add update_dependencies condarc option and --update-deps/--no-update-deps command line flags
- Allow specs with conda update --all
- Add --show-channel-urls and --no-show-channel-urls command line options
- Add always_copy condarc option
- Conda clean properly handles multiple envs directories. This breaks backwards compatibility with some of the --json output. Some of the old --json keys are kept for backwards compatibility.

4.148 3.17.0 (2015-09-11)

- Add windows_forward_slashes option to walk_prefix(), see #1513
- Add ability to set CONDA_FORCE_32BIT environment variable, it should should only be used when running conda-build, #1555
- Add config option to makes the Python dependency on pip optional, #1577
- Fix an UnboundLocalError
- Print note about pinned specs in no packages found error
- Allow wildcards in AND-connected version specs

- Print pinned specs to the debug log
- Fix conda create --clone with create_default_packages
- Give a better error when a proxy isn't found for a given scheme
- Enable running 'conda run' in offline mode
- Fix issue where hardlinked cache contents were being overwritten
- Correctly skip packages whose dependencies can't be found with conda update --all
- Use clearer terminology in -m help text.
- Use splitlines to break up multiple lines throughout the codebase
- Fix AttributeError with SSLError

4.149 3.16.0 (2015-08-10)

- Rename binstar -> Anaconda, see #1458
- Fix --use-local when the conda-bld directory doesn't exist
- Fixed --offline option when using "conda create --clone", see #1487
- Don't mask recursion depth errors
- Add conda search --reverse-dependency
- Check whether hardlinking is available before linking when using "Python install.py --link" directly, see #1490
- Don't exit nonzero when installing a package with no dependencies
- Check which features are installed in an environment via track_features, not features
- Set the verify flag directly on CondaSession (fixes conda skeleton not respecting the ssl_verify option)

4.150 3.15.1 (2015-07-23)

- Fix conda with older versions of argcomplete
- Restore the --force-pscheck option as a no-op for backwards compatibility

4.151 3.15.0 (2015-07-22)

- Sort the output of conda info package correctly
- Enable tab completion of conda command extensions using argcomplete. Command extensions that import conda should use conda.cli.conda_argparse.ArgumentParser instead of argparse.ArgumentParser. Otherwise, they should enable argcomplete completion manually.
- Allow psutil and pycosat to be updated in the root environment on Windows
- Remove all mentions of pscheck. The --force-pscheck flag has been removed.
- Added support for S3 channels
- Fix color issues from pip in conda list on Windows

- Add support for other machine types on Linux, in particular ppc64le
- Add non_x86_linux_machines set to config module
- Allow ssl_verify to accept strings in addition to boolean values in condarc
- Enable --set to work with both boolean and string values

4.152 3.14.1 (2015-06-29)

- Make use of Crypto.Signature.PKCS1_PSS module, see #1388
- Note when features are being used in the unsatisfiable hint

4.153 3.14.0 (2015-06-16)

- Add ability to verify signed packages, see #1343 (and conda-build #430)
- Fix issue when trying to add 'pip' dependency to old Python packages
- Provide option "conda info --unsafe-channels" for getting unobscured channel list, #1374

4.154 3.13.0 (2015-06-04)

- Avoid the Windows file lock by moving files to a trash directory, #1133
- Handle env dirs not existing in the Environments completer
- Rename binstar.org -> anaconda.org, see #1348
- Speed up 'source activate' by ~40%

4.155 3.12.0 (2015-05-05)

- Correctly allow conda to update itself
- Print which file leads to the "unable to remove file" error on Windows
- Add support for the no_proxy environment variable, #1171
- Add a much faster hint generation for unsatisfiable packages, which is now always enabled (previously it would not run if there were more than ten specs). The new hint only gives one set of conflicting packages, rather than all sets, so multiple passes may be necessary to fix such issues
- Conda extensions that import conda should use conda.cli.conda_argparser.ArgumentParser instead of argparse.ArgumentParser to conform to the conda help guidelines (e.g., all help messages should be capitalized with periods, and the options should be preceded by "Options:" for the sake of help2man).
- Add confirmation dialog to conda remove. Fixes conda remove --dry-run.

4.156 3.11.0 (2015-04-22)

- Fix issue where forced update on Windows could cause a package to break
- Remove detection of running processes that might conflict
- Deprecate --force-pscheck (now a no-op argument)
- Make conda search --outdated --names-only work, fixes #1252
- Handle the history file not having read or write permissions better
- Make multiple package resolutions warning easier to read
- Add --full-name to conda list
- Improvements to command help

4.157 2015-04-06 3.10.1:

- 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/<version> to HTTP header User-Agent string

4.158 2015-03-12 3.10.0:

- change default repo urls to be https
- Add --offline to conda search
- Add --names-only and --full-name to conda search
- Add tab completion for packages to conda search

4.159 2015-02-24 3.9.1:

- pscheck: check for processes in the current environment, see #1157
- don't write to the history file if nothing has changed, see #1148
- conda update --all installs packages without version restrictions (except for Python), see #1138
- conda update --all ignores the anaconda metapackage, see #1138
- use forward slashes for file urls on Windows
- don't symlink conda in the root environment from activate

- use the correct package name in the progress bar info
- use json progress bars for unsatisfiable dependencies hints
- don't let requests decode gz files when downloaded

4.160 2015-02-16 3.9.0:

- remove (de)activation scripts from conda, those are now in conda-env
- pip is now always added as a Python dependency
- allow conda to be installed into environments which start with _
- Add argcomplete tab completion for environments with the -n flag, and for package names with install, update, create, and remove

4.161 2015-02-03 3.8.4:

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

4.162 2015-01-28 3.8.3:

- simplified how ROOT_PREFIX is obtained in (de)activate

4.163 2015-01-27 3.8.2:

- Add conda clean --source-cache to clean the conda build source caches
- Add missing quotes in (de)activate.bat, fixes problem in Windows when conda is installed into a directory with spaces
- Fix conda install --copy

4.164 2015-01-23 3.8.1:

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

4.165 2015-01-22 3.8.0:

- move active script into conda-env, which is now a new dependency
- load the channel urls in the correct order when using concurrent.futures
- Add optional 'icondata' key to json files in conda-meta directory, which contain the base64 encoded png file or the icon
- remove a debug print statement

4.166 2014-12-18 3.7.4:

- Add --offline option to install, create, update and remove commands, and also add ability to set "offline: True" in condarc file
- Add conda uninstall as alias for conda remove
- Add conda info --root
- Add conda.pip module
- Fix CONDARC pointing to non-existing file, closes issue #961
- make update -f work if the package is already up-to-date
- Fix possible TypeError when printing an error message
- link packages in topologically sorted order (so that pre-link scripts can assume that the dependencies are installed)
- Add --copy flag to install
- prevent the progressbar from crashing conda when fetching in some situations

4.167 3.7.3 (2014-11-05)

- Conda install from a local conda package (or a tar fill which contains conda packages), will now also install the dependencies listed by the installed packages.
- Add SOURCE_DIR environment variable in pre-link subprocess
- Record all created environments in ~/.conda/environments.txt

4.168 3.7.2 (2014-10-31)

- Only show the binstar install message once
- Print the fetching repodata dot after the repodata is fetched
- Write the install and remove specs to the history file
- Add '-y' as an alias to '--yes'
- The *--file* option to conda config now defaults to os.environ.get('CONDARC')
- Some improvements to documentation (--help output)

- Add user_rc_path and sys_rc_path to conda info --json
- Cache the proxy username and password
- Avoid warning about conda in pscheck
- Make ~/.conda/envs the first user envs dir

4.169 3.7.1 (2014-10-07)

- Improve error message for forgetting to use source with activate and deactivate, see issue #601
- Don't allow to remove the current environment, see issue #639
- Don't fail if binstar_client can't be imported for other reasons, see issue #925
- Allow spaces to be contained in conda run
- Only show the conda install binstar hint if binstar is not installed
- Conda info package_spec now gives detailed info on packages. conda info path has been removed, as it is duplicated by conda package -w path.

4.170 3.7.0 (2014-09-19)

- Faster algorithm for --alt-hint
- Don't allow channel_alias with allow_other_channels: false if it is set in the system .condarc
- Don't show long "no packages found" error with update --all
- Automatically add the Binstar token to urls when the binstar client is installed and logged in
- Carefully avoid showing the binstar token or writing it to a file
- Be more careful in conda config about keys that are the wrong type
- Don't expect directories starting with conda- to be commands
- No longer recommend to run conda init after pip installing conda. A pip installed conda will now work without being initialized to create and manage other environments
- The rm function on Windows now works around access denied errors
- Fix channel urls now showing with conda list with show_channel_urls set to true

4.171 3.6.4 (2014-09-08)

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

4.172 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 Linux/macOS style path separators for prefixes in has_prefix on Windows (if the placeholder path uses /, replace it with a path that uses /, not \)
- Fix bug in --use-local due to API changes in conda-build
- Include user site directories in conda info -s
- Make binary has_prefix replacement work with spaces after the prefix
- Make binary has_prefix replacement replace multiple occurrences of the placeholder in the same null-terminated string
- Don't show packages from other platforms as installed or cached in conda search
- Be more careful about not warning about conda itself in pscheck
- Use a progress bar for the unsatisfiable packages hint generation
- Don't use TemporaryFile in try_write, as it is too slow when it fails
- Ignore InsecureRequestWarning when ssl_verify is False
- Conda remove removes features tracked by removed packages in track_features

4.173 3.6.2 (2014-08-20)

- Add --use-index-cache to conda remove
- Fix a bug where features (like mkl) would be selected incorrectly
- Use concurrent.future.ThreadPool to fetch package metadata asynchronously in Python 3.
- Do the retries in rm_rf on every platform
- Use a higher cutoff for package name misspellings
- Allow changing default channels in "system" .condarc

4.174 3.6.1 (2014-08-13)

- Add retries to download in fetch module
- Improved error messages for missing packages
- More robust rm_rf on Windows
- Print multiline help for subcommands correctly

4.175 3.6.0 (2014-08-11)

- Correctly check if a package can be hard-linked if it isn't extracted yet
- Change how the package plan is printed to better show what is new, updated, and downgraded
- Use suggest_normalized_version in the resolve module. Now versions like 1.0alpha that are not directly recognized by verlib's NormalizedVersion are supported better
- Conda run command, to run apps and commands from packages
- More complete --json API. Every conda command should fully support --json output now.
- Show the conda_build and requests versions in conda info
- Include packages from setup.py develop in conda list (with use_pip)
- Raise a warning instead of dying when the history file is invalid
- Use urllib.quote on the proxy password
- Make conda search --outdated --canonical work
- Pin the Python version during conda init
- Fix some metadata that is written for Python during conda init
- Allow comments in a pinned file
- Allow installing and updating menuinst on Windows
- Allow conda create with both --file and listed packages
- Better handling of some nonexistent packages
- Fix command line flags in conda package
- Fix a bug in the ftp adapter

4.176 3.5.5 (2014-06-10)

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

4.177 3.5.4 (2014-06-10)

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

4.178 3.5.3 (2014-06-09)

- Fix conda update to correctly not install packages that are already up-to-date
- Always fail with connection error in download
- The package resolution is now much faster and uses less memory
- Add ssl_verify option in condarc to allow ignoring SSL certificate verification, see issue #737

4.179 3.5.2 (2014-05-27)

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

4.180 3.5.1 (2014-05-26)

- Fix proxy support - conda now prompts for proxy username and password again
- Fix activate.bat on Windows with spaces in the path
- Update optional psutil dependency was updated to psutil 2.0 or higher

4.181 3.5.0 (2014-05-15)

- Replace use of urllib2 with requests. requests is now a hard dependency of conda.
- Add ability to only allow system-wise specified channels
- Hide binstar from output of conda info

4.182 3.4.3 (2014-05-05)

- Allow prefix replacement in binary files, see issue #710
- Check if creating hard link is possible and otherwise copy, during install
- Allow circular dependencies

4.183 3.4.2 (2014-04-21)

- Conda clean --lock: skip directories that don't exist, fixes #648
- Fixed empty history file causing crash, issue #644
- Remove timezone information from history file, fixes issue #651
- Fix PackagesNotFound error for missing recursive dependencies
- Change the default for adding cache from the local package cache - known is now the default and the option to use index metadata from the local package cache is --unknown
- Add --alt-hint as a method to get an alternate form of a hint for unsatisfiable packages
- Add conda package --ls-files to list files in a package
- Add ability to pin specs in an environment. To pin a spec, add a file called pinned to the environment's conda-meta directory with the specs to pin. Pinned specs are always kept installed, unless the --no-pin flag is used.
- Fix keyboard interrupting of external commands. Now keyboard interrupting conda build correctly removes the lock file
- Add no_link ability to conda, see issue #678

4.184 3.4.1 (2014-04-07)

- Always use a pkgs cache directory associated with an envs directory, even when using -p option with an arbitrary a prefix which is not inside an envs dir
- Add setting of PYTHONHOME to conda info --system
- Skip packages with bad metadata

4.185 3.4.0 (2014-04-02)

- Added revision history to each environment:
  - conda list --revisions
  - conda install --revision
  - log is stored in conda-meta/history
- Allow parsing pip-style requirement files with --file option and in command line arguments, e.g. conda install 'numpy>=1.7', issue #624
- Fix error message for --file option when file does not exist
- Allow DEFAULTS in CONDA_ENVS_PATH, which expands to the defaults settings, including the condarc file
- Don't install a package with a feature (like mkl) unless it is specifically requested (i.e., that feature is already enabled in that environment)
- Add ability to show channel URLs when displaying what is going to be downloaded by setting "show_channel_urls: True" in condarc
- Fix the --quiet option
- Skip packages that have dependencies that can't be found

4.186 3.3.2 (2014-03-24)

- Fix the --file option
- Check install arguments before fetching metadata
- Fix a printing glitch with the progress bars
- Give a better error message for conda clean with no arguments
- Don't include unknown packages when searching another platform

4.187 3.3.1 (2014-03-19)

- Fix setting of PS1 in activate.
- Add conda update --all.
- Allow setting CONDARC=' ' to use no condarc.
- Add conda clean --packages.
- Don't include bin/conda, bin/activate, or bin/deactivate in conda package.

4.188 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 Linux and macOS.
- On Linux and macOS, 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 Linux or macOS, you should run this command again, as the activate and deactivate scripts have changed.
- Begin work to support Python 3.4.
- Fix a bug in version comparison
- Fix usage of sys.stdout and sys.stderr in environments like pythonw on Windows where they are nonstandard file descriptors.

4.189 3.2.1 (2014-03-12)

- Fix installing packages with irrational versions
- Fix installation in the api
- Use a logging handler to print the dots

4.190 3.2.0 (2014-03-11)

- Print dots to the screen for progress
- Move logic functions from resolve to logic module

4.191 3.2.0a1 (2014-03-07)

- Conda now uses pseudo-boolean constraints in the SAT solver. This allows it to search for all versions at once, rather than only the latest (issue #491).
- Conda contains a brand new logic submodule for converting pseudo-boolean constraints into SAT clauses.

4.192 3.1.1 (2014-03-07)

- Check if directory exists, fixed issue #591

4.193 3.1.0 (2014-03-07)

- Local packages in cache are now added to the index, this may be disabled by using the --known option, which only makes conda use index metadata from the known remote channels
- Add --use-index-cache option to enable using cache of channel index files
- Fix ownership of files when installing as root on Linux
- Conda search: add '.' symbol for extracted (cached) packages

4.194 3.0.6 (2014-02-20)

- Fix 'conda update' taking build number into account

4.195 3.0.5 (2014-02-17)

- Allow packages from create_default_packages to be overridden from the command line
- Fixed typo install.py, issue #566
- Try to prevent accidentally installing into a non-root conda environment

4.196 3.0.4 (2014-02-14)

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

4.197 3.0.3 (2014-02-06)

- Improve the speed of clean --lock
- Some fixes to conda config
- More tests added
- Choose the first solution rather than the last when there are more than one, since this is more likely to be the one you want.

4.198 3.0.2 (2014-02-03)

- Fix detection of prefix being writable

4.199 3.0.1 (2014-01-31)

- Bug: not having track_features in condarc now uses default again
- Improved test suite
- Remove numpy version being treated special in plan module
- If the post-link.(bat|sh) fails, don't treat it as though it installed, i.e. it is not added to conda-meta
- Fix activate if CONDA_DEFAULT_ENV is invalid
- Fix conda config --get to work with list keys again
- Print the total download size
- Fix a bug that was preventing conda from working in Python 3
- Add ability to run pre-link script, issue #548

4.200 3.0.0 (2014-01-24)

- Removed build, convert, index, and skeleton commands, which are now part of the conda-build project: https://github.com/conda/conda-build
- Limited pip integration to *conda list*, that means *conda install* no longer calls *pip install* # !!!
- Add ability to call sub-commands named 'conda-x'
- The -c flag to conda search is now shorthand for --channel, not --canonical (this is to be consistent with other conda commands)
- Allow changing location of .condarc file using the CONDARC environment variable
- Conda search now shows the channel that the package comes from

- Conda search has a new --platform flag for searching for packages in other platforms.
- Remove condarc warnings: issue #526#issuecomment-33195012

4.201 2.3.1 (2014-01-17)

- Add ability create info/no_softlink
- Add conda convert command to convert non-platform-dependent packages from one platform to another (experimental)
- unify create, install, and update code. This adds many features to create and update that were previously only available to install. A backwards incompatible change is that conda create -f now means --force, not --file.

4.202 2.3.0 (2014-01-16)

- Automatically prepend http://conda.binstar.org/ (or the value of channel_alias in the .condarc file) to channels whenever the channel is not a URL or the word 'defaults or 'system'
- Recipes made with the skeleton pypi command will use setuptools instead of distribute
- Re-work the setuptools dependency and entry_point logic so that non console_script entry_points for packages with a dependency on setuptools will get correct build script with conda skeleton pypi
- Add -m, --mkdir option to conda install
- Add ability to disable soft-linking

4.203 2.2.8 (2014-01-06)

- Add check for chrpath (on Linux) before build is started, see issue #469
- Conda build: fixed ELF headers not being recognized on Python 3
- Fixed issues: #467, #476

4.204 2.2.7 (2014-01-02)

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

4.205 2.2.6 (2013-12-31)

- Fix test section for automatic recipe creation from pypi using --build-recipe
- Minor Py3k fixes for conda build on Linux
- Copy symlinks as symlinks, issue #437
- Fix explicit install (e.g. from output of *conda list -e*) in root env
- Add pyyaml to the list of packages which can not be removed from root environment
- Fixed minor issues: #365, #453

4.206 2.2.5 (2013-12-17)

- Conda build: move broken packages to conda-bld/broken
- Conda config: automatically Add the 'defaults' channel
- Conda build: improve error handling for invalid recipe directory
- Add ability to set build string, issue #425
- Fix LD_RUN_PATH not being set on Linux under Python 3, see issue #427, thanks peter1000

4.207 2.2.4 (2013-12-10)

- Add support for execution with the -m switch (issue #398), i.e. you can execute conda also as: Python -m conda
- Add a deactivate script for windows
- Conda build adds .pth-file when it encounters an egg (TODO)
- Add ability to preserve egg directory when building using build/preserve_egg_dir: True
- Allow track_features in ~/.condarc
- Allow arbitrary source, issue #405
- Fixed minor issues: #393, #402, #409, #413

4.208 2.2.3 (2013-12-03)

- Add "foreign mode", i.e. disallow install of certain packages when using a "foreign" Python, such as the system Python
- Remove activate/deactivate from source tarball created by sdist.sh, in order to not overwrite activate script from virtualenvwrapper

4.209 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, .<name>-post-link.sh and .<name>-pre-unlink.sh

4.210 2.2.1 (2013-11-15)

- Minor fixes related to make conda pip installable
- Generated conda meta-data missing 'files' key, fixed issue #357

4.211 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 Linux and macOS
- Add 'use_pip' to condarc configuration file
- Fixed activate on Windows to set CONDA_DEFAULT_ENV
- Allow setting "always_yes: True" in condarc file, which implies always using the --yes option whenever asked to proceed

4.212 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 <root>/conda-recipes) from pypi then build a conda package (and install it) * Pip list and pip install only happen if pip is installed
- Enhance the locking mechanism so that conda can call itself in the same process.

4.213 2.0.4 (2013-11-04)

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

4.214 2.0.3 (2013-10-28)

- Update bundle format
- Fix bug when displaying packages to be downloaded (thanks Crystal)

4.215 2.0.2 (2013-10-27)

- Add --index-cache option to clean command, see issue #321
- Use RPATH (instead of RUNPATH) when building packages on Linux

4.216 2.0.1 (2013-10-23)

- Add --no-prompt option to conda skeleton pypi
- Add create_default_packages to condarc (and --no-default-packages option to create command)

4.217 2.0.0 (2013-10-01)

- Added user/root mode and ability to soft-link across filesystems
- Added create --clone option for copying local environments
- Fixed behavior when installing into an environment which does not exist yet, i.e. an error occurs
- Fixed install --no-deps option
- Added --export option to list command
- Allow building of packages in "user mode"
- Regular environment locations now used for build and test
- Add ability to disallow specification names
- Add ability to read help messages from a file when install location is RO
- Restore backwards compatibility of share/clone for conda-api
- Add new conda bundle command and format
- Pass ARCH environment variable to build scripts
- Added progress bar to source download for conda build, issue #230
- Added ability to use url instead of local file to conda install --file and conda create --file options

4.218 1.9.1 (2013-09-06)

- Fix bug in new caching of repodata index

4.219 1.9.0 (2013-09-05)

- Add caching of repodata index
- Add activate command on Windows
- Add conda package --which option, closes issue 163
- Add ability to install file which contains multiple packages, issue 256
- Move conda share functionality to conda package --share
- Update documentation
- Improve error messages when external dependencies are unavailable
- Add implementation for issue 194: post-link or pre-unlink may append to a special file ${PREFIX}/.messages.txt for messages, which is display to the user's console after conda completes all actions

- Add conda search --outdated option, which lists only installed packages for which newer versions are available
- Fixed numerous Py3k issues, in particular with the build command

4.220 1.8.2 (2013-08-16)

- Add conda build --check option
- Add conda clean --lock option
- Fixed error in recipe causing conda traceback, issue 158
- Fixes conda build error in Python 3, issue 238
- Improve error message when test command fails, as well as issue 229
- Disable Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- Simplified locking, in particular locking should never crash conda when files cannot be created due to permission problems

4.221 1.8.1 (2013-08-07)

- Fixed conda update for no arguments, issue 237
- Fix setting prefix before calling should_do_win_subprocess() part of issue 235
- Add basic subversion support when building
- Add --output option to conda build

4.222 1.8.0 (2013-07-31)

- Add Python 3 support (thanks almarklein)
- Add Mercurial support when building from source (thanks delicb)
- Allow Python (and other packages which are used by conda itself) to be updated in root environment on Windows
- Add conda config command
- Add conda clean command
- Removed the conda pip command
- Improve locking to be finer grained
- Made activate/deactivate work with zsh (thanks to mika-fischer)
- Allow conda build to take tarballs containing a recipe as arguments
- Add PKG_CONFIG_PATH to build environment variables
- Fix entry point scripts pointing to wrong Python when building Python 3 packages
- Allow source/sha1 in meta.yaml, issue 196
- More informative message when there are unsatisfiable package specifications

- Ability to set the proxy urls in condarc
- Conda build asks to upload to binstar. This can also be configured by changing binstar_upload in condarc.
- Basic tab completion if the argcomplete package is installed and eval "$(register-Python-argcomplete conda)" is added to the bash profile.

4.223 1.7.2 (2013-07-02)

- Fixed conda update when packages include a post-link step which was caused by subprocess being lazily imported, fixed by 0d0b860
- Improve error message when 'chrpath' or 'patch' is not installed and needed by build framework
- Fixed sharing/cloning being broken (issue 179)
- Add the string LOCKERROR to the conda lock error message

4.224 1.7.1 (2013-06-21)

- Fix "executable" not being found on Windows when ending with .bat when launching application
- Give a better error message from when a repository does not exist

4.225 1.7.0 (2013-06-20)

- Allow ${PREFIX} in app_entry
- Add binstar upload information after conda build finishes

4.226 1.7.0a2 (2013-06-20)

- Add global conda lock file for only allowing one instance of conda to run at the same time
- Add conda skeleton command to create recipes from PyPI
- Add ability to run post-link and pre-unlink script

4.227 1.7.0a1 (2013-06-13)

- Add ability to build conda packages from "recipes", using the conda build command, for some examples, see: https://github.com/ContinuumIO/conda-recipes
- Fixed bug in conda install --force
- Conda update command no longer uses anaconda as default package name
- Add proxy support
- Added application API to conda.api module
- Add -c/--channel and --override-channels flags (issue 121).

- Add default and system meta-channels, for use in .condarc and with -c (issue 122).
- Fixed ability to install ipython=0.13.0 (issue 130)

4.228 1.6.0 (2013-06-05)

- Update package command to reflect changes in repodata
- Fixed refactoring bugs in share/clone
- Warn when Anaconda processes are running on install in Windows (should fix most permissions errors on Windows)

4.229 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 no longer updates to Python 3, also ensure that conda itself is always installed into the root environment (issue 110)

4.230 1.6.0rc1 (2013-05-30)

- Major internal refactoring
- Use new "depends" key in repodata
- Uses pycosat to solve constraints more efficiently
- Add hard-linking on Windows
- Fixed linking across filesystems (issue 103)
- Add conda remove --features option
- Added more tests, in particular for new dependency resolver
- Add internal DSL to perform install actions
- Add package size to download preview
- Add conda install --force and --no-deps options
- Fixed conda help command
- Add conda remove --all option for removing entire environment
- Fixed source activate on systems where sourcing a gives "bash" as $0
- Add information about installed versions to conda search command
- Removed known "locations"
- Add output about installed packages when update and install do nothing
- Changed default when prompted for y/n in CLI to yes

4.231 1.5.2 (2013-04-29)

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

4.232 1.5.1 (2013-04-19)

- Fixed issue 71 and (73 duplicate): not being able to install packages starting with conda (such as 'conda-api')
- Fixed issue 69 (not being able to update Python / NumPy)
- Fixed issue 76 (cannot install mkl on OSX)

4.233 1.5.0 (2013-03-22)

- Add conda share and clone commands
- Add (hidden) --output-json option to clone, share and info commands to support the conda-api package
- Add repo sub-directory type 'linux-armv6l'

4.234 1.4.6 (2013-03-12)

- Fixed channel selection (issue #56)

4.235 1.4.5 (2013-03-11)

- Fix issue #53 with install for meta packages
- Add -q/--quiet option to update command

4.236 1.4.4 (2013-03-09)

- Use numpy 1.7 as default on all platfroms

4.237 1.4.3 (2013-03-09)

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

4.238 1.4.2 (2013-03-08)

- Feature selection fix for update
- Windows: don't allow linking or unlinking Python from the root environment because the file lock, see issue #42

4.239 1.4.1 (2013-03-07)

- Fix some feature selection bugs
- Never exit in activate and deactivate
- Improve help and error messages

4.240 1.4.0 (2013-03-05)

- Fixed conda pip NAME==VERSION
- Added conda info --license option
- Add source activate and deactivate commands
- Rename the old activate and deactivate to link and unlink
- Add ability for environments to track "features"
- Add ability to distinguish conda build packages from Anaconda packages by adding a "file_hash" meta-data field in info/index.json
- Add conda.builder.share module

4.241 1.3.5 (2013-02-05)

- Fixed detecting untracked files on Windows
- Removed backwards compatibility to conda 1.0 version

4.242 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 if no condarc file exists (and license is valid)

4.243 1.3.3 (2013-01-23)

- Fix conda create not filtering channels correctly
- Remove (hidden) --test and --testgui options

4.244 1.3.2 (2013-01-23)

- Fix deactivation of packages with same build number note that conda upgrade did not suffer from this problem, as was using separate logic

4.245 1.3.1 (2013-01-22)

- Fix bug in conda update not installing new dependencies

4.246 1.3.0 (2013-01-22)

- Added conda package command
- Added conda index command
- Added -c, --canonical option to list and search commands
- Fixed conda --version on Windows
- Add this changelog

4.247 1.2.1 (2012-11-21)

- Remove ambiguity from conda update command

4.248 1.2.0 (2012-11-20)

- `conda upgrade` now updates from AnacondaCE to Anaconda (removed upgrade2pro)
- Add versioneer

4.249 1.1.0 (2012-11-13)

- Many new features implemented by Bryan

4.250 1.0.0 (2012-09-06)

- Initial release

CHAPTER

FIVE

COMMAND REFERENCE

- *Conda general commands*
- *Conda vs. pip vs. virtualenv commands*

Conda provides many commands for managing packages and environments. The links on this page provide help for each command. You can also access help from the command line with the `--help` flag:

```
conda install --help
```

5.1 Conda general commands

The following commands are part of conda:

5.1.1 `conda clean`

Remove unused packages and caches.

Options:

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

Removal Targets

-a, --all Remove index cache, lock files, unused cache packages, and tarballs.

-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. Argument is path(s) to prefix(es) where files should be found and removed.

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 Do not ask for confirmation.

Examples:

conda clean --tarballs

5.1.2 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.

Options:

```
usage: conda config [-h] [--json] [-v] [-q] [--system | --env | --file FILE]
                    [--show [SHOW [SHOW ...]] | --show-sources | --validate |
                    --describe [DESCRIBE [DESCRIBE ...]] | --write-default]
                    [--get [KEY [KEY ...]] | --append KEY VALUE | --prepend
                    KEY VALUE | --set KEY VALUE | --remove KEY VALUE |
                    --remove-key KEY | --stdin]
```

Output, Prompt, and Flow Control Options

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

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

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

Config File Location Selection

Without one of these flags, the user config file at '/home/docs/.condarc' is used.

--system Write to the system .condarc file at '/home/docs/checkouts/readthedocs.org/user_builds/continuumio-conda/conda/latest/.condarc'.

--env Write to the active conda environment .condarc file (<no active environment>). If no environment is active, write to the user config file (/home/docs/.condarc).

--file Write to the given file.

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.
--describe	Describe given configuration parameters. If no arguments given, show information for all configuration parameters.
--write-default	Write the default configuration to a file. Equivalent to *conda config --describe > ~/.condarc*.

Config Modifiers

--get	Get a configuration value.
--append	Add one configuration value to the end of a list key.
--prepend, --add	Add one configuration value to the beginning of a list key.
--set	Set a boolean or string key
--remove	**Remove a configuration value from a list key. This removes** all instances of the value.
--remove-key	Remove a configuration key (and all its values).
--stdin	Apply configuration information given in yaml format piped through stdin.

See *conda config --describe* or https://conda.io/docs/config.html for details on all the options that can go in .condarc.

Examples:

Display all configuration values as calculated and compiled:

```
conda config --show
```

Display all identified configuration sources:

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

Describe all available configuration options:

```
conda config --describe
```

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
```

5.1.3 conda create

Create a new conda environment from a list of specified packages. To use the created environment, use 'conda activate envname' look in that directory first. This command requires either the -n NAME or -p PREFIX option.

Options:

```
usage: conda create [-h] [--clone ENV] [-n ENVIRONMENT | -p PATH] [-c CHANNEL]
                    [--use-local] [--override-channels]
                    [--repodata-fn REPODATA_FNS] [--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] [--dev]
                    [package_spec [package_spec ...]]
```

Positional Arguments

package_spec Packages to install or update in the conda environment.

Named Arguments

--clone Path to (or name of) 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 source against old Python versions.

Target Environment Specification

-n, --name Name of environment.

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

Channel Customization

-c, --channel **Additional channel to search for packages. These are URLs searched in the order** they are given (including file:// for local directories). 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 http://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 name of repodata on remote server. 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 reduced in time scope. You may pass this flagmore than once. Leftmost entries are tried first, and the fallback torepodata.json is added for you automatically.

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.

Package Linking and Install-time Options

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

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

-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 Do not ask for confirmation.

--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 create -n myenv sqlite
```

5.1.4 conda info

Display information about current conda install.

Options:

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

Named Arguments

-a, --all	Show all information.
--base	Display base environment path.
-e, --envs	List all known conda environments.
-s, --system	List environment variables.
--unsafe-channels	Display list of channels with tokens exposed.

Output, Prompt, and Flow Control Options

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

5.1.5 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] [--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]
```

(continues on next page)

(continued from previous page)

```
                    [--file FILE] [--force-reinstall]
                    [--freeze-installed | --update-deps | -S | --update-all | --
↪update-specs]
                    [-m] [--clobber] [--dev]
                    [package_spec [package_spec ...]]
```

Positional Arguments

package_spec Packages to install or update in the conda environment.

Named Arguments

--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 source against old Python versions.

Target Environment Specification

-n, --name Name of environment.

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

Channel Customization

-c, --channel **Additional channel to search for packages. These are URLs searched in the order** they are given (including file:// for local directories). 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 http://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 name of repodata on remote server. 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 reduced in time scope. You may pass this flagmore than once. Leftmost entries are tried first, and the fallback torepodata.json is added for you automatically.

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.

--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.

-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'.

--update-all, --all Update all installed packages in the environment.

--update-specs Update based on provided specifications.

Package Linking and Install-time Options

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

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

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

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

-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 Do not ask for confirmation.

--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 install -n myenv scipy

5.1.6 `conda list`

List linked 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]
```

Positional Arguments

regex List only packages matching this regular expression.

Named Arguments

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

-c, --canonical Output canonical names of packages only. Implies --no-pip.

-f, --full-name Only search for full names, i.e., ^<regex>$.

--explicit List explicitly all installed conda packaged with URL (output may be used by conda create --file).

--md5 Add MD5 hashsum when using --explicit

-e, --export Output requirement string only (output may be used by conda create --file).

-r, --revisions List the revision history and exit.

--no-pip Do not include pip-only installed packages.

Target Environment Specification

-n, --name Name of environment.

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

Output, Prompt, and Flow Control Options

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

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

5.1.7 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]
```

Named Arguments

-w, --which	Given some 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	Package name of the created package.
--pkg-version	Package version of the created package.
--pkg-build	Package build number of the created package.

Target Environment Specification

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

5.1.8 conda remove

Remove a list of packages from a specified conda environment.

> 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] [--all]
                    [--features] [--force-remove] [--no-pin] [-C] [-k]
                    [--offline] [-d] [--json] [-q] [-v] [-y] [--dev]
                    [package_name [package_name ...]]
```

Positional Arguments

package_name Package names to remove from the environment.

Named Arguments

--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 source against old Python versions.

Target Environment Specification

-n, --name Name of environment.

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

Channel Customization

-c, --channel **Additional channel to search for packages. These are URLs searched in the order** they are given (including file:// for local directories). 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 http://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 name of repodata on remote server. 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 reduced in time scope. You may pass this flagmore than once. Leftmost entries are tried first, and the fallback torepodata.json is added for you automatically.

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 file.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

-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 Do not ask for confirmation.

Examples:

conda remove -n myenv scipy

5.1.9 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] [-C] [-k] [--offline]
                    [--json] [-v] [-q]
```

Named Arguments

--envs Search all of the current user's environments. If run as Administrator (on Windows) or UID 0 (on unix), search all known environments on the system.

-i, --info Provide detailed information about each package.

--subdir, --platform Search the given subdir. Should be formatted like 'osx-64', 'linux-32', 'win-64', and so on. The default is to search the current platform.

Channel Customization

-c, --channel **Additional channel to search for packages. These are URLs searched in the order** they are given (including file:// for local directories). 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 http://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 name of repodata on remote server. 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 reduced in time scope. You may pass this flagmore than once. Leftmost entries are tried first, and the fallback torepodata.json is added for you automatically.

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

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

-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]'

5.1.10 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]
                    [--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] [--force-reinstall]
                    [--freeze-installed | --update-deps | -S | --update-all | --
↪update-specs]
                    [--clobber]
                    [package_spec [package_spec ...]]
```

Positional Arguments

package_spec Packages to install or update in the conda environment.

Named Arguments

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

Target Environment Specification

-n, --name Name of environment.

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

Channel Customization

-c, --channel **Additional channel to search for packages. These are URLs searched in the order** they are given (including file:// for local directories). 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 http://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 name of repodata on remote server. 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 reduced in time scope. You may pass this flagmore than once. Leftmost entries are tried first, and the fallback torepodata.json is added for you automatically.

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.

--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.

-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'.

--update-all, --all Update all installed packages in the environment.

--update-specs Update based on provided specifications.

Package Linking and Install-time Options

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

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

Networking Options

-C, --use-index-cache Use cache of channel index files, even if it has expired.

-k, --insecure Allow conda to perform "insecure" SSL connections and transfers. Equivalent to setting 'ssl_verify' to 'false'.

--offline Offline mode. Don't connect to the Internet.

Output, Prompt, and Flow Control Options

-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 Do not ask for confirmation.

--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
```

5.2 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

SIX

GLOSSARY

- *.condarc*
- *Activate/Deactivate environment*
- *Anaconda*
- *Anaconda Cloud*
- *Anaconda Navigator*
- *Channels*
- *conda*
- *conda environment*
- *conda package*
- *conda repository*
- *Metapackage*
- *Miniconda*
- *Noarch package*
- *Package manager*
- *Packages*
- *Repository*
- *Silent mode installation*

6.1 .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".

6.2 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.

6.3 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*.

6.4 Anaconda Cloud

A web-based, repository hosting service in the cloud. Packages created locally can be published to the cloud to be shared with others. Anaconda Cloud is a public version of Anaconda Repository.

6.5 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.

6.6 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."

6.7 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.

6.8 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`

6.9 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.

6.10 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/

6.11 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.

6.12 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*.

6.13 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".

6.14 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.

6.15 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.

6.16 Repository

Any storage location from which software assets may be retrieved and installed on a local computer. See also *Anaconda Cloud* and *conda repository*.

6.17 Silent mode installation

When installing Miniconda or Anaconda in silent mode, screen prompts are not shown on screen and default settings are automatically accepted.

PYTHON MODULE INDEX

C

conda.api, 126
conda.cli.python_api, 125
conda.core.solve, 123

INDEX

C

CLEAN (*conda.cli.python_api.Commands attribute*), 125
Commands (*class in conda.cli.python_api*), 125
conda.api (*module*), 126
conda.cli.python_api (*module*), 125
conda.core.solve (*module*), 123
CONFIG (*conda.cli.python_api.Commands attribute*), 125
CREATE (*conda.cli.python_api.Commands attribute*), 125

D

DepsModifier (*class in conda.core.solve*), 123

F

first_writable() (*conda.api.PackageCacheData static method*), 128

G

get() (*conda.api.PackageCacheData method*), 129
get() (*conda.api.PrefixData method*), 130

H

HELP (*conda.cli.python_api.Commands attribute*), 125

I

INFO (*conda.cli.python_api.Commands attribute*), 125
INSTALL (*conda.cli.python_api.Commands attribute*), 125
is_writable() (*conda.api.PackageCacheData property*), 129
is_writable() (*conda.api.PrefixData property*), 130
iter_records() (*conda.api.PackageCacheData method*), 129
iter_records() (*conda.api.PrefixData method*), 130
iter_records() (*conda.api.SubdirData method*), 128

L

LIST (*conda.cli.python_api.Commands attribute*), 125

N

NO_DEPS (*conda.core.solve.DepsModifier attribute*), 123
NOT_SET (*conda.core.solve.DepsModifier attribute*), 123

O

ONLY_DEPS (*conda.core.solve.DepsModifier attribute*), 123

P

PackageCacheData (*class in conda.api*), 128
PrefixData (*class in conda.api*), 130

Q

query() (*conda.api.PackageCacheData method*), 129
query() (*conda.api.PrefixData method*), 130
query() (*conda.api.SubdirData method*), 128
query_all() (*conda.api.PackageCacheData static method*), 129
query_all() (*conda.api.SubdirData static method*), 128

R

reload() (*conda.api.PackageCacheData method*), 129
reload() (*conda.api.PrefixData method*), 130
reload() (*conda.api.SubdirData method*), 128
REMOVE (*conda.cli.python_api.Commands attribute*), 125
RUN (*conda.cli.python_api.Commands attribute*), 125
run_command() (*in module conda.cli.python_api*), 125

S

SEARCH (*conda.cli.python_api.Commands attribute*), 125
solve_final_state() (*conda.api.Solver method*), 126
solve_final_state() (*conda.core.solve.Solver method*), 123
solve_for_diff() (*conda.api.Solver method*), 127

solve_for_diff() (*conda.core.solve.Solver method*), 124
solve_for_transaction() (*conda.api.Solver method*), 127
solve_for_transaction() (*conda.core.solve.Solver method*), 124
Solver (*class in conda.api*), 126
Solver (*class in conda.core.solve*), 123
SubdirData (*class in conda.api*), 127

U

UPDATE (*conda.cli.python_api.Commands attribute*), 125