The Definitive Guide to Conda Environments

How to manage environments with conda for Python & R.

Matthew Sarmiento Follow
May 23, 2019 · 10 min read ★

Conda's natural environment. Illustration by Johann Wenzel Peter.

Conda environments are like cousins of Python's virtual environments. Both serve to help manage dependencies and isolate projects, and they function in a similar way, with one key distinction: conda environments are **language agnostic**. That is, they support languages other than Python.

🪀 In this guide we'll cover the basics of creating and managing environments with `conda` for Python & R.

⚠️ **Note:** We'll be using the latest versions of Conda v4.6.x, Python v3.7.y, and R v3.5.z on macOS Mojave throughout this guide.

Table of Contents

. . .

Conda vs. Pip vs. Venv — What's the Difference?

Before we get started, some of you might be wondering what the difference is between `conda`, `pip`, and `venv`.

I'm glad you asked. We can't put it any better than this: `pip` is a package manager *for Python*. `venv` is an environment manager *for Python*. `conda` is **both a package and environment manager** and is **language agnostic**.

Whereas `venv` creates isolated environments for Python development only, `conda` can create isolated environments for any language (in theory).

Whereas `pip` only installs Python packages from PyPI, `conda` can both

- Install packages (written in any language) from repositories like Anaconda Repository and Anaconda Cloud.
- Install packages from PyPI by using `pip` in an active Conda environment.

How cool is that?

👉🏽 For a handy, if slightly outdated, chart comparing the three, click here (don't forget to scroll right!).

. . .

Morning Mist by Thomas Cole.

Using Conda Environments

Creating Environments

To create an environment with `conda` for Python development, run:

```
% conda create --name conda-env python                    # Or use -n
```

💥 **Important:** Replace "conda-env" with the name of your environment. From here on we'll always use "conda-env" for the name of our environments.

This environment will use the same version of Python as your current shell's Python interpreter. To specify a different version of Python, use:

```
% conda create -n conda-env python=3.7
```

You can also install additional packages when creating an environment, like, say, `numpy` and `requests`.

```
% conda create -n conda-env numpy requests
```

⚠️ **Note:** Because `conda` ensures dependencies are satisfied when installing packages, Python will be installed alongside `numpy` and `requests` 😄.

You can also specify which versions of packages you'd like to install.

```
% conda create -n conda-env python=3.7 numpy=1.16.1 requests=2.19.1
```

⚠️ **Note:** It's recommended to install all the packages you want to include in an environment at the same time to help avoid dependency conflicts.

Last, you can activate your environment with the invocation:

```
% conda activate conda-env
(conda-env) %                          # Fancy new command prompt
```

And deactivate it with:

```
% conda deactivate
%                                    # Old familiar command prompt
```

. . .

Where Environments Live

When you create an environment with Python's `venv` module, you need to say where it lives by specifying its path.

```
% python3 -m venv /path/to/new/environment
```

Environments created with `conda`, on the other hand, live by default in the `envs/` folder of your Conda directory, whose path will look something like:

```
% /Users/user-name/miniconda3/envs          # Or .../anaconda3/envs
```

I prefer the approach taken by `venv` for two reasons.

1️⃣ It makes it easy to tell if a project utilizes an isolated environment by including the environment as a sub-directory.

```
my-project/
├── conda-env                      # Project uses an isolated env ✅
├── data
├── src
└── tests
```

2️⃣ It allows you to use the same name for all of your environments (I use "conda-env"), meaning you can activate each with the same command.

```
% cd my-project/
% conda activate conda-env
```

💸 **Bonus:** This allows you to alias the activation command and stick it in your `.bashrc` file, making life a little bit simpler.

⚠️ **Note:** If you keep all of your environments in your Conda's `env/` folder, you'll have to give each of them a different name, which can be a pain 😞.

So, how *do* you place environments outside of your Conda's `env/` folder? By using the `--prefix` flag instead of `--name` when creating an environment.

```
% conda create --prefix /path/to/conda-env              # Or use -p
```

⚠️ **Note:** This makes an environment called "conda-env" in the specified path.

It's as simple as that. *However*, placing environments outside of the default `env/` folder comes with two drawbacks.

1️⃣ `conda` can no longer find your environment with the `--name` flag. Instead, you'll need to pass the `--prefix` flag along with the environment's full path. For example, when installing packages, which we'll cover in the next section.

2️⃣ Your command prompt is no longer prefixed with the active environment's name, but rather its full path.

```
(/path/to/conda-env) %
```

As you can imagine, this gets messy quickly. Like this doozy, for instance.

```
(/Users/user-name/data-science/project-name/conda-env) %        # 😨
```

Fortunately, there's an easy fix. You just need to modify the `env_prompt` setting in your `.condarc` file, which you can do with a single stroke.

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

⚠️ **Note:** This will edit your `.condarc` file if you already have one and create one if you do not. For more on modifying your `.condarc` file, see the docs.

Now your command prompt will only display the active environment's name.

```
% conda activate /path/to/conda-env
(conda-env) %                                          # Woohoo! 🎉
```

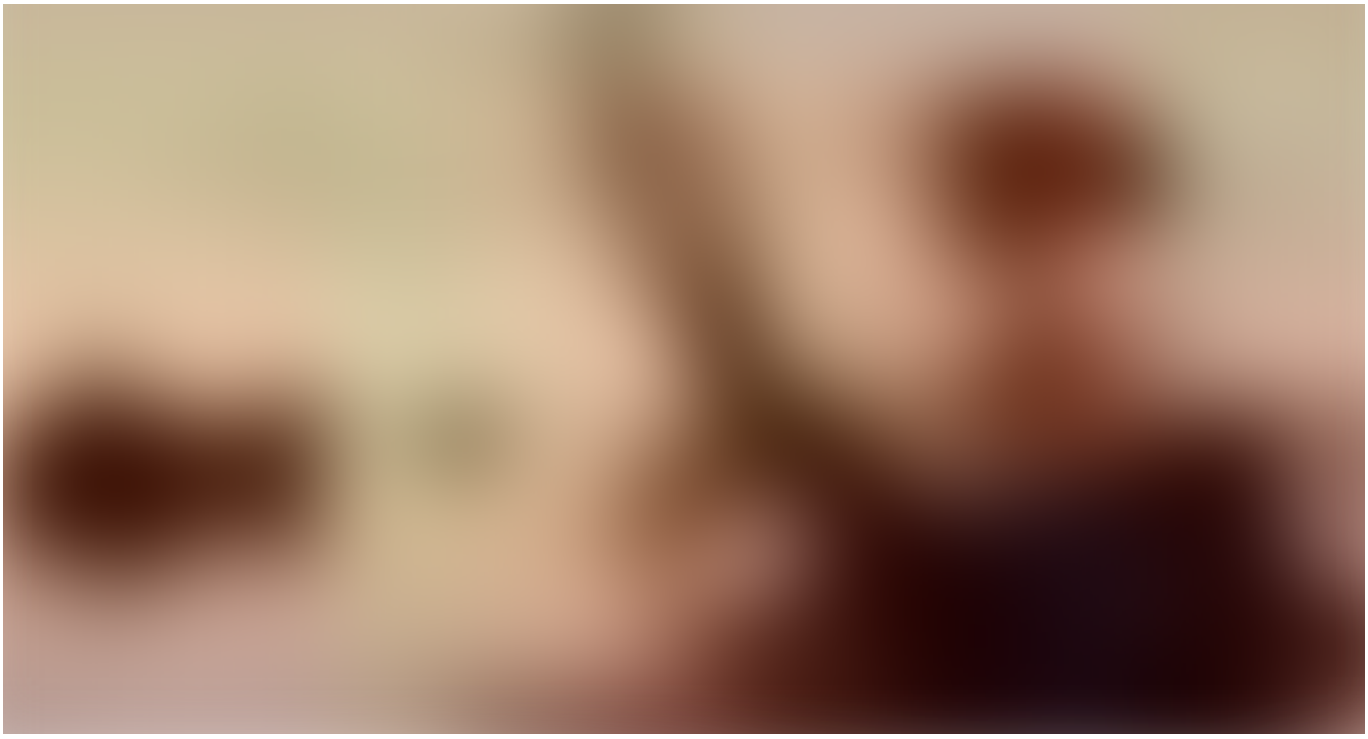
Last, you can view a list of all your existing environments.

```
% conda env list

# conda environments:
#
                         /path/to/conda-env
base                  *  /Users/username/miniconda3
r-env                    /Users/username/miniconda3/envs/r-env
```

⚠️ **Note:** The `*` points to the current active environment. A bit annoyingly though, it will point to "base" even when no environment is active 🤷.

. . .



An American Lake Scene by Thomas Cole.

Installing Packages

There are two ways to install packages with `conda`.

1️⃣ From inside an active environment.

2️⃣ From your default shell.

The latter requires you to point to the environment you want to install packages in using the same flag (`--name` or `--prefix`) that you used to create your environment with.

The former works equally well regardless of which flag you used.

💥 **Important:** We *strongly* recommend sticking to the former approach, as it removes the danger of unintentionally installing packages system-wide.

♻️ **Reminder:** All environments in this guide are named “conda-env”. You can replace “conda-env” with the name of your environment.

• • •

From the Anaconda Repository

By default, `conda` installs packages from Anaconda Repository. Once you've created an environment, you can install additional packages in two ways.

1️⃣ From inside an active environment.

```
(conda-env) % conda install pandas=0.24.1                    # 🐼
```

2️⃣ From your default shell.

```
% conda install -n conda-env pandas=0.24.1      # Or -p /path/to/env
```

Likewise, you can update the packages in an environment in two ways.

1️⃣ From inside the active environment.

```
(conda-env) % conda update pandas
```

2️⃣ From your default shell.

```
% conda update -n conda-env pandas             # Or -p /path/to/env
```

You can also list the packages installed in a given environment in — yep, you guessed it — two ways.

1️⃣ From inside the active environment.

```
(conda-env) % conda list
```

2️⃣ From your default shell.

```
% conda list -n conda-env                    # Or -p /path/to/env
```

. . .

From Other Conda Repositories

If a package isn't available from the default Anaconda Repository, you can try searching for it on Anaconda Cloud, which hosts Conda packages provided by third party repositories like Conda-Forge.

To install a package from Anaconda Cloud, you'll need to use the `--channel` flag to specify the repository you want to install from. For example, if you wanted to install `opencv` from Conda-Forge, you'd run:

```
(conda-env) % conda install --channel conda-forge opencv    # Or -c
```

Thankfully, `conda` keeps track of where a package was installed from.

```
(conda-env) % conda list

# packages in environment at /path/to/conda-env:
#
# Name                    Version                   Build             Channel

numpy                     1.16.1           py37h926163e_0
opencv                    4.1.0            py37h0cb0d9f_3    conda-forge
pandas                    0.24.2           py37h0a44026_0
```

The blank channel entries for `numpy` and `pandas` represent the `default_channels`, which is set by default to Anaconda Repository.

⚠️ **Note:** For brevity we've shown only a selection of packages above.

You can also permanently add a channel as a package source.

```
% conda config --append channels conda-forge
```

This will modify your `.condarc` file to look something like this:

```
env_prompt: '({name}) '      # Modifies active environment prompt
channels:                    # Lists package sources to install from
- defaults                   # Default Anaconda Repository
- conda-forge
```

🚨 **Caution:** The order of your channels *matters*. If a package is available from multiple channels, `conda` will install it from the channel listed *highest* in your `.condarc` file. For more on managing channels, see the docs.

. . .

From PyPI

If a package isn't available from Anaconda Repository or Anaconda Cloud, you can try installing it with `pip`, which `conda` installs by default in any environment created with Python.

For example, to install requests with `pip` you'd run:

```
(conda-env) % pip install requests
```

Note that `conda` correctly lists PyPI as the channel for `requests`, making it easy to identify packages installed with `pip`.

```
(conda-env) % conda list

# packages in environment at /path/to/conda-env:
#
# Name                    Version                   Build  Channel
```

```
numpy                1.16.1           py37h926163e_0
opencv               4.1.0            py37h0cb0d9f_3    conda-forge
pandas               0.24.2           py37h0a44026_0
requests             2.21.0                  pypi_0    pypi
```

🚨 **Caution:** As `pip` packages do not possess all the features of `conda` packages, it's strongly recommended to install packages with `conda` whenever possible. For more on `conda` vs. `pip` packages, click here.

. . .

Moonlight by Thomas Cole.

Managing Environments

Environment Files

The easiest way to make your work reproducible by others is to include a file in your project's root directory listing all the packages, along with their version numbers, that are installed in your project's environment.

Conda calls these environment files. They are the exact analogue of requirements files for Python's virtual environments.

Like with everything else, you can make an environment file in two ways.

1️⃣ From inside an active environment.

```
(conda-env) % conda env export --file environment.yml       # Or -f
```

2️⃣ From your default shell.

```
% conda env export -n conda-env -f /path/to/environment.yml
```

Your `environment.yml` file will look something like this:

```
name: null                          # Our env was made with --prefix
channels:
  - conda-forge                     # We added a third party channel
  - defaults
dependencies:
  - numpy=1.16.3=py37h926163e_0
  - opencv=3.4.2=py37h6fd60c2_1
  - pandas=0.24.2=py37h0a44026_0
  - pip=19.1.1=py37_0
  - pip:                            # Packages installed from PyPI
    - requests==2.21.0
prefix: /Users/user-name/data-science/project-name/conda-env
```

⚠️ **Note:** For brevity we've shown only a selection of packages above.

• • •

Duplicating Environments

Given an `environment.yml` file, you can easily recreate an environment.

```
% conda env create -n conda-env -f /path/to/environment.yml
```

💸 **Bonus:** You can also add the packages listed in an `environment.yml` file to an existing environment with:

```
% conda env update -n conda-env -f /path/to/environment.yml
```

. . .

View in the White Mountains by Thomas Cole.

Environments With R

To use R in an environment, all you need to do is install the `r-base` package.

```
(conda-env) % conda install r-base
```

Of course, you can always do this when first creating an environment.

```
% conda create -n r-env r-base
```

⚠️ **Note:** Replace "r-env" with the name of your environment.

Conda's R packages are available from the R channel of Anaconda Cloud, which is included by default in Conda's `default_channels` list, so you don't need to specify the R channel when installing R packages like, say, `tidyverse`.

```
% conda activate r-env
(r-env) % conda install r-tidyverse
```

⚠️ **Note:** All packages from the R channel are prefixed with "`r-`".

If you want, you can install the `r-essentials` bundle, which includes over 80 of the most popular scientific R packages, like `tidyverse` and `shiny`.

```
(r-env) % conda install r-essentials
```

Last, if you want to install an R package that Conda doesn't offer, you'll need to build the package from CRAN, instructions to which you can find here.

. . .

Further Reading

If by chance you find yourself wondering how it is Conda environments work exactly, check out this blurb on how Python's virtual environments do their thing. Conda environments work in exactly the same fashion.

Other than that, that about does it for us. If you'd like to stay up to date with my data science-y postings, feel free to follow me on twitter.

Cheers, and happy reading.

. . .

Update 08/2019: Conda Revisions

You really do learn something new everyday. This morning my friend, Kumar Shishir, told me about yet another **incredibly useful** `conda` feature: conda revisions.

I couldn't believe my ears. How could I have languished for so long in complete and utter ignorance of such a brilliant feature?

Revisions track changes to your environment over time, allowing you to **easily** remove packages **and** all of their dependencies.

For example, say we created a new `conda-env` and installed `numpy`, followed by `pandas`. Our revision history would look like this:

```
(conda-env) % conda list --revisions

2019-08-30 16:04:14  (rev 0)              # Created our env

+pip-19.2.2
+python-3.7.4

2019-08-30 16:04:30  (rev 1)              # Installed numpy

+numpy-1.16.4
+numpy-base-1.16.4

2019-08-30 16:04:39  (rev 2)              # Installed pandas
```

```
+pandas-0.25.1
+python-dateutil-2.8.0
+pytz-2019.2
```

Imagine we no longer wanted to have `pandas` in our environment because it's (somehow) incompatible with our earlier dependencies, or because we simply no longer need it.

Revisions allow us to rollback our environment to a previous incarnation:

```
(conda-env) % conda install --revision 1
(conda-env) % conda list --revisions         # (Showing latest only)

2019-08-30 16:08:05  (rev 3)                 # Uninstalled pandas

-pandas-0.25.1
-python-dateutil-2.8.0
-pytz-2019.2
```

The – signs by each package tell us we've successfully removed them from our environment. Now we're ready to get back to some data science 😎.

. . .

Update 02/2020: Clear Your Tarballs!

As you build more projects, each with their own environment, you'll begin to quickly accumulate `tarballs` from packages you've installed.

To get rid of them and free up some disc space, run:

```
% conda clean --all                    # no active env needed
```