How to Setup a Python Virtual Environment

Julie Perilla Garcia Follow

Jan 7, 2020 · 5 min read ★

Photo by Shahadat Rahman on Unsplash

If you are a Python programmer, this has probably happened to you. You have just spent hours installing the packages that your script needs to run, only to find out that you broke some other piece of code that you previously wrote. You upgraded a package that you need for your current script, and the new version does not work with your old script.

This issue is not unique to Python. In my Windows days, we called this “DLL hell.” The problem of managing packages and versions is a problem that dates back to when programmers started packaging code.

Most programming languages now have elegant ways of dealing with this. In Python, virtual environments are the standard. This guide will help you to quickly set up a new project in Python and keep its packages isolated from the rest of your code using virtualenv and pip. These instructions assume you are on a Mac. For instructions on how to do this on windows, see here.

Install Homebrew

If you aren’t already using Homebrew on your Mac, you should consider it. Homebrew is a package manager for macOS that makes the installation of tools that you need globally really easy. It installs each package inside of the Homebrew dedicated folder and then links each package to /usr/local using a symlink. For full instructions on how to install Homebrew, go here. But it is pretty simple.

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Install Python

MacOS comes with Python 2.7 installed. If you are writing code for any modern script or application, you will probably want to use Python 3. Primarily because Python 2 will no longer be supported after January 1, 2020. Be aware that Python 3 is not backward compatible, so any scripts you have written in Python 2 will need to be updated or run with the Python 2 interpreter.

This section will help you to install python 3.7 and pip3, which is the standard Python package manager. Open a terminal and run the following command.

```
brew install python
```

To test if Python and pip installed correctly, run the following commands.

```
python3 --version
pip3 --version
```

At this point, you should be able to run Python 3.7 interactively from the command line.

```
python3
```

If you see a >>, it worked. You can use this interactive terminal to test out lines of code in Python before putting them into an actual script or project. To get out of this state, use Ctrl+D or type exit().

Install Virtualenv

Virtualenv is a tool that will allow you to keep each of your Python projects in a neat isolated environment that won't affect your other projects. To install virtualenv globally, run the following command. You will reuse virtualenv every time you create a new project.

```
pip3 install -U virtualenv
```

Create Your Virtual Environment

First, you will need to create a directory for your new project.

```
mkdir new-python-project
cd new-python-project
```

Next, create your virtual environment using the following command.

```
virtualenv venv
```

This command will create a folder inside of your project's root directory that will hold all of your installed packages. (Don't forget to add this folder to .gitingore if you are checking code into a git repository.)

Your venv folder will now hold a local version of Python 3 and Python 2.7 and a local version of pip and pip2. We will demonstrate below that the python and pip commands will now point to the local Python 3 and pip3 versions, only while the virtual environment is activated.

Activate Your Virtual Environment

This command gets your virtual environment up and running.

```
source venv/bin/activate
```

You should now see the "(venv)" indicator at the beginning of your command prompt.

```
(venv) prompt$
```

Now try running the **which** command to see what installations of Python and pip will run by default. This command should show you the path to your local venv versions of Python and pip.

```
which python
which pip
```

At any time, you can stop your virtual environment by running the deactivate command.

```
(venv) prompt$ deactivate
prompt$
```

Then try running the **which** command again. Now, which should default back to using the globally installed versions in /usr/local. Be aware that, when the virtual environment is not active, any packages installed with the **pip** command will install globally.

```
which python
which pip
```

So, inside the activated virtual environments, we can use the **python** and **pip** commands for Python 3, and we will not need **python3** and **pip3**.

Install Packages

To demonstrate the power of **Virtualenv**, we will install a package called arrow that makes date and time manipulation easy. If your virtual environment is activated, as we did in the previous step, the **pip** command will install all packages locally to the venv folder (and nowhere else!). Doing so will help you to avoid installing endless packages globally and having to manage several versions for different projects.

```
# Make sure your venv is activated
source venv/bin/activate

# Install the arrow package locally
pip install -U arrow

# Check what packages are now installed locally
pip list
```

To search for other Python packages you might want to use, go the Python Package Index (PyPi) is a comprehensive resource.

Create a Python File and Run

Now we will write our Python script. Create a file called run.py in the root directory of your project.

```
touch run.py
```

Open the file in your favorite text editor and add the following code.

```
#!/usr/bin/python
import time
import arrow

utc = arrow.utcnow()
print(f"UTC time is = {utc}")

pacific = utc.to('US/Pacific')
print(f"Pacific time is = {pacific}")

birth_date = arrow.get('You were born in February 21, 1990', 'MMMM D,
YYYY')
print(f"Your birth date is {birth_date}")
print(f"You were born {birth_date.humanize()}")
```

Save the file. Run your code. Make sure your virtual environment is running first.

```
source venv/bin/activate
python run.py
```

Stop your virtual environment.

```
deactivate
```

Now try running your code.

```
python run.py
```

You should get an error at this point saying “ModuleNotFoundError: No module named ‘arrow.’” This happens because **arrow** was installed inside of the virtual environment and is not accessible when the venv is not running. To run your script, first type the activate command above.

(Optional) Make it Reusable

If you are checking your project into source control and want someone to be able to recreate this environment on their machine, or you want to run your code on a server, a simple way to do this is to create a **requirements.txt** file. This file tells pip and other developers what Python packages your project depends on to run. This command will copy all of the packages and current versions into a file called requirements.txt. You can expect to see this file in most Python projects.

```
# Copies all packages currently installed into the file
pip freeze > requirements.txt
```

Now that you have everything packaged nicely, you can check your code into a source code repository like GitHub and try pulling it to a different environment. Using git, this is how another developer could easily recreate your environment and run your code on their machine. Create a file called READme.md in the root directory of your project and add the following instructions.

```
git clone https://github.com/username/your-awesome-code.git
cd your-awesome-code
source venv/bin/activate
pip install -r requirements.txt
python run.py
```

That’s it. Happy coding!

Python Programming Technology Software Development Virtualenv