pyenv, virtualenv and using them with Jupyter

python pyenv virtualenv jupyter

Aug 17, 2020

It is common that the different projects you are working on depend on **different versions of Python**. That is why **pyenv** becomes very handy for Python developers, as it lets you switch between different Python versions easily. With **pyenv-virtualenv** it can also be used together with **virtualenv** to create isolated development environments for different projects with different dependencies.

For example, if some of the projects you are working on requires Tensorflow 1.15, while your system's Python is of version 3.8, you must find some ways to install Python 3.7 in order to work on your project, as Tensorflow 1.15 can only be run in Python 3.5 to Python 3.7.

This article aims at giving a quick introduction to pyenv and pyenv-virtualenv, as well as describing how one can easily create new kernels of virtual environments in **Jupyter**.

Installing and Using pyenv

pyenv works on macOS and Linux, but not Windows (except inside the Windows Subsystem for Linux). Windows users might want to check out **pyenv-win** for further information.

On macOS, it can be installed using Homebrew:

```
$ brew update
$ brew install pyenv
```

On both macOS and Linux, it can also be installed by checking out the latest version of pyenv. For details of installing pyenv this way, refer to the offical installation guidelines here: **https://github.com/pyenv/pyenv#installation**.

After installation, add the following line to your `.bashrc` (or `.zshrc`) file:

```
eval "$(pyenv init -)"
```

Once you have pyenv installed, you can do a few things like below:

Installing a Python version

```
# List all available Python versions
$ pyenv install --list

# Install a specific Python version (3.7.8)
$ pyenv install 3.7.8

# List Python version installed
$ pyenv versions
* system (set by /Users/....)
  3.7.8
```

Setting a local Python version

```
# Set the Python version for the current directory
$ pyenv local 3.7.8

# Now by default you will be using Python 3.7.8
$ python
Python 3.7.8 (default, Aug 17 2020, 11:05:21)
>>>

# Unset it and change back to system default
$ pyenv local --unset
```

Setting a global Python version

```
# Install a new version and set it as system default
$ pyenv install 2.7.6
$ pyenv global 2.7.6

# Now you have 2.7.6 as the default Python version
$ python
Python 2.7.6 (default, Aug 17 2020, 11:08:23)
>>>
```

Using virtualenv with pyenv

pyenv by itself only allows you to switch between different Python versions. To create an isolated environment with a set of dependencies, we will need **virtualenv** too. You can follow the steps below to set up your computer to use pyenv and virtualenv together.

Firstly, we need ot install virtualenv:

```
$ pip3 install virtualenv
$ pip3 install virtualenvwrapper
```

Next, we need to install pyenv-virtualenv. This can be done on macOS by using brew as follows (or follow the instructions on **this page** if you are not using macOS):

```
$ brew install pyenv-virtualenv
```

Finally, add the following line to your `.bashrc` or `.zshrc` file:

```
eval "$(pyenv virtualenv-init -)"
```

Once you are done with the steps above, you can create new virtual environments as follows:

```
# Install a new Python version
$ pyenv install 3.7.4

# Create a new virtualenv named myenv with Python 3.7.4
$ pyenv virtualenv 3.7.4 tf1.15

# Go to the project directory, and set its local environment
$ cd ~/repo/my-project
$ pyenv local tf1.15

# Install dependencies as needed
$ pip3 install tensorflow==1.15
```

Adding Kernels to Jupyter

It is also common that we use Jupyter for quick prototyping and testing. It would be convenient if we can invoke different virtual environments in Jupyter to test our source codes. In fact, it is very easy to create new kernels of different virtual environments in Jupyter.

Firstly, you have to check the paths of your Juypyter installation. (Note that it does not matter which environment you are using to run your Jupyter notebook or Jupyter lab.) You can check the paths using the following command:

```
$ jupyter --paths
```

On my computer, it is something like below. What we need to note here is the `data` path.

```
config:
    /Users/albert/.jupyter
    /usr/local/Cellar/python@3.8/3.8.4/Frameworks/Python.framework/Versions/3.8/etc/jupyter
    /usr/local/etc/jupyter
    /etc/jupyter
data:
    /Users/albert/Library/Jupyter
    /usr/local/Cellar/python@3.8/3.8.4/Frameworks/Python.framework/Versions/3.8/share/jupyter
    /usr/local/share/jupyter
    /usr/share/jupyter
runtime:
    /Users/ayeung/Library/Jupyter/runtime
```

Next, we will need to check the path to the Python interpreter of the virtual environment:

```
# Activate your virtualenv
$ pyenv activate tf1.15

# Check path of the Python interpreter
$ pyenv which python
/Users/albert/.pyenv/versions/tf1.15/bin/python

# Deactivate the virtualenv
$ pyenv deactivate
```

Finally, we create a new folder under the `kernels` directory:

```
$ mkdir /User/albert/Library/Jupyter/kernels/tf1.15
```

and add a new file named `kernel.json` in that directory with the following content:

```
{
  "argv": [
    "/User/albert/.pyenv/versions/tf1.15/bin/python",
    "-m", "ipykernel",
    "-f", "{connection_file}"
  ],
  "display_name": "tf1.15",
  "language": "python"
}
```

Once this is done, you will be able to use the kernel in Jupyter.

PREVIOUS
BERT - Tokenization and Encoding