Using and Installing Python

Ocelote

There are three versions of Python available on the new cluster. The naming convention is different from the older clusters to support the latest version which is 3. Python version 3 requires the `python3` command or `pip3 list` to differentiate. It is very different from Python version 2, so do not assume that Python 3 will work for you or that all older modules will work with version 3.

1. Python 2.6.6 comes with the operating system. `python --version` will show this level. You will generally not use this one.
2. Python 3.5.5 is the default version and can be loaded with `module load python/3.5`. There are many packages installed shown by `pip3 list`.
3. Python 3.6.5 is available but has to be loaded with `module load python/3.6`. This version contains many of the machine learning packages like Tensorflow that can be utilized on the Centos 7 / GPU nodes.
4. Python 2.7.14 is available as a module but will need to be invoked with `module load python/2` or `module load python/2.7`. Python 2 is no longer officially supported by the Python Software Foundation.

ElGato

There are two versions of Python available on ElGato.

1. Python 2.7.5 is the default if you just want to use the system version
2. Python 3.5.5 is provided with `module load python/3.5`

Python 3.5 has the basic packages installed. `pip3 list` will display the packages along with the version installed.

You are encouraged to use virtualenv to customize Python (below).

Python Package Policy

We maintain a two tiered approach to Python packages

- **tier 1:** We install the basic Python packages that are required by most users (these are mostly libraries rather than packages, such as numpy and scipy). This is done for the versions of Python that we install as modules. Adding some packages might force an upgrade of numpy for example, which might break a user's environment that was dependent on the prior version.

- **tier 2:** For packages that we do not provide we STRONGLY recommend the use of virtualenv, which is detailed below and provides a custom and easy to use person Python environment.

Installing Python Packages Using virtualenv

Useful overview of virtualenv and venv

InfoWorld Article: Python virtualenv and venv do's and don'ts

One of the best things about Python is the number of packages provided by the user community. On a personal machine, the most popular method today for managing these packages is the use of a package manager, like pip. Unfortunately, these require root access and are not a viable solution on the clusters.

There is an easy solution. You can use virtualenv to create a personal python environment that will persist for each time you log in. There is no risk of packages being updated under you for another user.

On Ocelote, virtualenv is set up for Python 3.5, and for Python 2. On Elgato, virtualenv is set up for Python 3.

To find packages you might want to start with `python.org`.

1. Set up your virtual environment. This is done one time only and will be good for all future uses of your Python environment.
   a. `module load python`, for example `python/2` or `python/3` on Ocelote
   b. `virtualenv --system-site-packages path` where path is where you want your python environment. You can use "~" for your home directory.
2. Implement the change
a. `source path/bin/activate` where path is the same as step 1.b
b. Append this line to your `.bashrc` if you want it to take effect on login. And add the `module load python` like in step 1.a. Be aware that this python profile will load every time you log in so you are likely to get errors if you try to use a different python version for something else.

3. At this point you can `pip install` a package to your new environment for example `pip install pycurl`

4. When you log out and return later, your installed package will still be present. If you do not do step 2.b you will have to do 2.a each time you log in.

### Accessing Custom Packages from a Jupyter Session

We don't recommend you try to install packages from within an OOD Jupyter session directly. Instead, to access custom packages from a Jupyter OOD session, install them in your local directory first. You will then need to show Jupyter where to find them. An example is shown below:

First, a virtual environment is set up nested in the user's `/extra/` directory:

```bash
$ module load python/3.5
$ mkdir /extra/NetID/python_example
$ virtualenv --system-site-packages /extra/NetID/python_example
$ source /extra/NetID/python_example/bin/activate
```

Next, install your custom package. In this case, installing the non-standard library `pycurl` using pip:

```bash
$ pip install pycurl
```

Then, in a Jupyter notebook session, point to where the package was locally installed (in this particular case):

```python
import sys
sys.path.append("/extra/NetID/python_example/share/doc")  # where pycurl is located
import pycurl # success!
```

Sometimes packages can wind up in different places, so you'll want to double-check the package's location if Jupyter is struggling to find it.

When using conda, it's the same general idea.