Using and Installing Python

Overview

Different versions of Python are available on the clusters. Python 2 is available on all of them but is no longer supported by the Python Foundation, so we recommend you use Python 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.

We recommend you use the virtual environment supported by Python, with more details below. One of the biggest reasons is that you can preserve your compute environment. The packages provided in the system Python are subject to change which may introduce incompatibilities at runtime.

Anaconda

You may prefer to work with Anaconda. It is also available as a module and the `conda` commands are available like `conda list` or `conda create` which is used to create an Anaconda environment similar to Python.

Ocelote

There are four versions of Python available on Ocelote.

<table>
<thead>
<tr>
<th>Version</th>
<th>Accessibility</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python 2.6.6</td>
<td>system version (no module)</td>
<td>Accessible as <code>python</code> when python 2.7.14 is not loaded as a module</td>
</tr>
<tr>
<td>Python 2.7.14</td>
<td>module load python:2 or module load python:2.7</td>
<td>Overrides system python 2.6.6</td>
</tr>
<tr>
<td>Python 3.5.5</td>
<td>module load python:3 or module load python:3.5</td>
<td></td>
</tr>
<tr>
<td>Python 3.6.5</td>
<td>module load python:3.6</td>
<td>Loaded by default with <code>module load python</code>. This version contains many of the machine learning packages like Tensorflow that can be utilized on the Centos 7 / GPU nodes.</td>
</tr>
</tbody>
</table>

ElGato
There are three versions of Python available on ElGato.

<table>
<thead>
<tr>
<th>Version</th>
<th>Accessibility</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python 2.7.5</td>
<td>system version (not a module)</td>
<td>Accessible as python</td>
</tr>
<tr>
<td>Python 3.6.8</td>
<td>system version (not a module)</td>
<td>Accessible as python3</td>
</tr>
<tr>
<td>Python 3.5.5</td>
<td>module load python/3.5</td>
<td>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).</td>
</tr>
<tr>
<td>Python 3.8.0</td>
<td>module load python/3.8.0</td>
<td>Includes many packages</td>
</tr>
</tbody>
</table>

Puma

Four versions of Python are available on Puma. Only available on compute nodes accessible either in a batch submission or interactive session.

<table>
<thead>
<tr>
<th>Version</th>
<th>Accessibility</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python 2.7.5</td>
<td>system version (no module)</td>
<td>Accessible as python</td>
</tr>
<tr>
<td>Python 3.6.8</td>
<td>system version (no module)</td>
<td>Accessible as python3 (unless python module is loaded)</td>
</tr>
<tr>
<td>Python 3.6.5</td>
<td>module load python/3.6/3.6.5</td>
<td>Includes many packages</td>
</tr>
<tr>
<td>Python 3.8.2</td>
<td>module load python/3.8/3.8.2</td>
<td>Includes more packages</td>
</tr>
</tbody>
</table>

⚠️ Python 2 is no longer officially supported by the Python Software Foundation.

Installation & 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 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 Python 3.6, and for Python 2.
On EIGato, virtualenv is set up for the python modules.
On Puma virtualenv is available for the python modules.
To find packages you might want to start with python.org.

Virtual Environment Instructions

1. Set up your virtual environment in your account. This step is done one time only and will be good for all future uses of your Python environment.

   Note: In the commands below, /path/to/virtual/env is the path to the directory where all of your environment’s executables and packages will be saved. For example, if you use the path ~/mypyenv, this will create a directory in your home called mypyenv. Inside will be directories bin, lib, lib64, and include.

   **Commands**
   **Python Version < 3.8**

   ```bash
   $ module load python/<version>
   $ virtualenv --system-site-packages /path/to/virtual/env
   
   Python Version 3.8
   
   $ module load python/<version>
   $ python3 -m venv /path/to/virtual/env
   
   2. To use your new environment, you’ll need to activate it. Inside your virtual environment, there’s a directory called bin that has a file called activate. Sourcing this will add all of the paths needed to your working environment. To activate, run the following, replacing /path/to/virtual/env with the path specific to your account:

   ```bash
   $ source /path/to/virtual/env/bin/activate
   
   3. Once your environment is active, you can pip install your package. For example:

   ```bash
   $ pip install pycurl
   
   4. If you would like your virtual environment to always be active, you can add the activate command to your ~/.bashrc. This is a hidden file in your home directory that sets up your environment each time you log in. To edit it, open the file using your favorite text editor. Then, add the following to a blank line:

   ```bash
   module load python/<version>
   source /path/to/virtual/env/bin/activate
   
   Installing Python Packages Using Conda

   Users can use conda to install packages locally in their account. For a cheat sheet on conda commands, see: https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html

   Example for setting up a local conda environment:

   ```bash
   $ module load anaconda/2020
   $ conda create --name py37 python=3.7 # Build a local environment with a specific version of python
   $ conda init bash # only needs to be run once
   $ conda activate py37 # activate your environment.
   
   Once your environment is activated, you will be able to download and use custom packages using conda.

   It should be noted that the conda init step will modify your ~/.bashrc file which may cause some issues when using HPC resources such as OOD. For information, see: FAQ -- resolving Anaconda issues.

Jupyter Notebooks on OOD
An OOD Jupyter session is a different environment from working directly on the command line. This means to access and install custom packages, you will need a slightly different approach. To execute system commands, precede them with a !

See here for more information on OpenOnDemand (OOD)

Creating a virtual environment

If you need to create a new virtual environment, you may do so with the following (substituting in your own name and path).

This step does not need to be performed if you already have a virtual environment in your account. For more information on creating a virtual environment, see above under Virtual Environment Instructions.

```bash
virtualenv --system-site-packages /path/to/virtualenv
```

Next, to install your custom package you'll need to activate your environment and perform the install. This must be done on the same line because system commands do not carry over from one line to the next.

```bash
source /path/to/virtualenv/bin/activate && pip install pycurl && pip show pycurl
```

The pip show command is used to get the location of where the package was installed. You'll need this because the install will not automatically allow access to the package, meaning if you try to import it in the next step you'll get an import/module not found error. The next step will allow you to add the location to your python system path

```python
import sys
sys.path.append("/path/to/virtualenv/lib/python3.5/site-packages") # where pycurl is located in this example
import pycurl # success!
```

You can always use pip show with any virtual environments in your account allowing you to find and import anything you've previously installed. Just start with the source command above and exclude the pip install (you may want to double-check that the python virtual environment and the Jupyter version match before using/importing packages).

Another option to get a path automatically added to your environment is to include an export PYTHONPATH statement in your ~/.bashrc. Be aware, though, that this adds the path to your environment every time you log in, so you may want to proceed with caution if you're switching between python versions:

```bash
export PYTHONPATH=/path/to/virtualenv/lib/python3.5/site-packages:$PYTHONPATH
```