Using Python in Compute Canada Clusters

So far we have seen how to work on the console, and a bit on how to create simple scripts. In this section we will be showing how to use Python programs in our clusters.

https://docs.computecanada.ca/wiki/Python (https://docs.computecanada.ca/wiki/Python)

Forget about Anaconda/Miniconda

It is tempting to install and use `conda` environments. For personal use (your own laptops) or single user (or just a few) it is OK and an important resource. For an HPC environment, however, it does not work well since Anaconda:

1. Often installs software already on our systems, with a configuration that is not optimal.
2. Makes incorrect assumptions about the location of various system libraries.
3. Anaconda uses the `$HOME` directory for its installation, where it writes an enormous number of files (think of your quota!).

1. Is slower than the installation of packages via Python wheels.
2. Modifies the `$HOME/.bashrc` file, which can easily cause conflicts.

Load the Python version required

By defaut our Python version is 2.7.13. This is likely to change in the future, but you can check our python versions by:

```
[user@gra-login3 ~]$ module spider python
```

```
------------------------------------------------------------------------------
------------
  python:
------------------------------------------------------------------------------
------------
    Description:
      Python is a programming language that lets you work more quickly and integrate
your
      systems more effectively.

     Versions:
        python/2.7.14
        python/3.5.4
        python/3.6.3
        python/3.7.0
        python/3.7.4
        python/3.8.0
        python/3.8.2
     Other possible modules matches:
        ipython-kernel  python27-mpi4py  python27-scipy-stack  python35-mpi4py  pytho
n35-scipy-stack

------------------------------------------------------------------------------
------------
  To find other possible module matches execute:

      $ module -r spider '.*python.*'

------------------------------------------------------------------------------
------------
  For detailed information about a specific "python" package (including how to load t
he modules) use the module's full name.
  Note that names that have a trailing (E) are extensions provided by other modules.
  For example:

     $ module spider python/3.8.2
------------------------------------------------------------------------------
------------
```

Once you identify the version you wish, you can load it by:

```
[user@gra-login3 ~]$ module load python/3.6.3
```

If you use `module load python` without any version, our systems will use the newest `python/3.8.2` by default. If you are going to develop your own code, we reccomend you avoid python 2 completely, and do all your coding in Python 3, as Python 2 is no longer supported.

Loading python modules in our systems will also load the `pip` installer so you can install particular python pakages or modules available in PyPi. However, **DO NOT** install packages we provide through:

1. Software stack: by `module` command
2. Python wheels: by `avail_wheels` command

Through the sofware stack we provide numerical and scientific Python pakages such as `scipy`, `numpy`, `pandas`, etc, bundled in a module prefixed with `scipy`:

```
[user@gra-login3 ~]$ module spider scipy



----------------------------------------------------------------------------
------------
  scipy-stack:
----------------------------------------------------------------------------
------------
    Description:
      Bundle which contains the Scientific Python stack, including Cycler, mpmath, nu
mpy, scipy,
      sympy, pandas, matplotlib, ipython_genutils, traitlets, ptyprocess, pathlib2, p
ickleshare,
      pexpect, simplegeneric, ipython, ipykernel, jupyter_client, jupyter_core, pyzm
q, tornado,
      futures and ipyparallel.

     Versions:
        scipy-stack/2017b
        scipy-stack/2018b
        scipy-stack/2019a
        scipy-stack/2019b

----------------------------------------------------------------------------
------------
  For detailed information about a specific "scipy-stack" package (including how to l
oad the modules) use the module's full name.
  Note that names that have a trailing (E) are extensions provided by other modules.
  For example:

     $ module spider scipy-stack/2019b
----------------------------------------------------------------------------
------------
```

Other packages are predownloaded in our clusters and can be found using the `avail_wheels` command:

```
[user@gra-login3 ~]$ avail_wheels msprime*
name     version    build    python    arch
-------  ---------  -------  --------  -------
msprime  0.7.3               cp38      generic
msprime  0.7.3               cp37      generic
msprime  0.7.3               cp36      generic
msprime  0.7.3               cp35      generic
```

Once you found the package and version, you can install it using `pip` with the `--no-index` option. This will avoid downloading the package from the web and use our optimized copies instead:

```
pip install --no-index msprime==0.7.3
```

In the login nodes you can download a personal copy from the web by `pip install <package name> --user` if you cannot find it in our wheels. However, **WE DO NOT RECOMMEND IT**, and we prefer if you create a ticket by send an email to ***support@computecanada.ca***.

NOTE: You will not be able to install packages globally (e.i. without the `--user` option)

Don't use `conda`, use a virtual environment instead

We provide the tool virtualenv to allows users to create virtual environments within which you can easily install Python packages and have a clean Python environment. After you loaded python as before, you can create your virtual environment by:

```
[user@gra-login3 ~]$ virtualenv --no-download ~/ENV`
[user@gra-login3 ~]$ source ~/ENV/bin/activate
(ENV)[user@gra-login3 ~]$ pip install --no-index --upgrade pip
```

On your Compute Canada account (or your guest account), try creating a virtual envronment and install the package `msprime`

Creating virtual environments inside of your jobs

The most efficient way to execute python code in our systems is creating a virtual environment within your job, and pointing at a local hard-drive:

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --mem-per-cpu=1.5G
#SBATCH --time=1:00:00
module load python/3.6
virtualenv --no-download $SLURM_TMPDIR/env
source $SLURM_TMPDIR/env/bin/activate
pip install --no-index --upgrade pip
pip install --no-index -r requirements.txt
python ...
```

Note the `$SLURM_TMPDIR` environmental variable that holds the path to the temporary directory. This directory is on the local hardisk, and accessing it is faster than `project` , `$HOME` , or `scratch`

The `requirements.txt` file is a regular text file with one package/requirement/dependency per line, including the version (otherwise the most recent is the default). Say that your python script requires `scikit_learn` version 0.23.0, and `astropy` version greater than 3.2.3:

```
# contents of requirements.txt file
scikit_learn==0.23.0
astropy>=3.2.3
```

Exercise

In this exercise we will run a demographic simulator package called `msprime` on the compute nodes.

> The primary goal of msprime is to efficiently and conveniently generate coalescent trees for a sample under a range of evolutionary scenarios

The basic command line we will be executings is `msp simulate -L 1000000  1000000 test.tsf` , that is, to simulate one million bases, sample one million individuls and output the tree sequence to `test.tsf`

Steps:

1. Write a slurm script called `myjob.sh` to submit a job with a python environment. **HINT:** check https://docs.computecanada.ca/wiki/Python#Creating_and_using_a_virtual_environment (https://docs.computecanada.ca/wiki/Python#Creating_and_using_a_virtual_environment)
2. Within that script, install msprime
3. Within that script, excute msprime: `msp simulate -L 1000000  1000000 test.tsf`
4. Submit the job through `sbatch myjob.sh`