Python Virtual Environments

This page describes the recommended setup for using Python virtual environments (or virtualenv) on the clusters.

A virtualenv is like a sandbox where one can install Python packages from public registries without affecting the default environment. Each environment resides in a self-contained directory, so multiple virtualenvs can co-exist side by side with different versions of tools or dependencies installed.

- Before starting
- Creating a virtualenv
- Using a virtualenv
- Installing new packages
- Stop using a virtualenv
- Removing a virtualenv
- Are you looking for pip3?

Before starting

Before you start it's necessary to choose which Python version you want to use. As is described on Using software on the SCITAS clusters, we have multiple versions of Python available, compiled with different compilers.

At the minimum one needs to choose between Python 2 and Python 3. You can see the versions available using module spider. In this example we first simply search for python and then search for a specific version, that lists which modules can be loaded to give us access to the version we are looking for. Below we choose Python version 3.7.3 compiled with GCC version 7.4.0 (these are in fact the default Python compiled with the default GCC).

### Choosing a Python version

```bash
[user@ucluster:~]$ module spider python
-------------------------------------------------------------------------
python:                                                                
-------------------------------------------------------------------------
Versions:
    python/2.7.16  
    python/3.7.3

[user@cluster:~]$ module spider python/3.7.3
-------------------------------------------------------------------------
python: python/3.7.3
-------------------------------------------------------------------------

You will need to load all module(s) on any one of the lines below before the "python/3.7.3" module is available to load.

    gcc/7.4.0
    gcc/8.3.0
    intel/18.0.5

[user@cluster:~]$ module load gcc/7.4.0 python/3.7.3
[user@cluster:~]$ python --version
Python 3.7.3
```

Creating a virtualenv

We can now create a virtualenv, the name can be whatever one wants but must not exist yet.
Creating a virtualenv

[user@cluster:~]$ virtualenv --system-site-packages venvs/venv-for-demo
Using base prefix '/ssoft/spack/humagne/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-7.4.0/python-3.7.3-S1m3vikrg4nq4tjhx76dgqy7zbt4k'f
New python executable in /home/user/venvs/venv-for-demo/bin/python3.7
Also creating executable in /home/user/venvs/venv-for-demo/bin/python
Installing setuptools, pip, wheel... done.

We strongly recommend that, as in the example above, one uses the --system-site-packages option. This ensures that the optimized packages already installed in the python modules are used.

Using a virtualenv

To use a virtualenv it's necessary to activate it:

Activating a virtualenv

[user@cluster:~]$ source venvs/venv-for-demo/bin/activate
(venv-for-demo) [user@cluster:~]$ module list
Currently Loaded Modules:
  1) gcc/7.4.0  2) python/3.7.3
(venv-for-demo) [user@cluster:~]$ pip install --no-cache-dir biopython

Once activated the prompt changes and any Python related commands that follow will refer to the Python installation and packages contained/linked in the virtualenv.

If all the packages in the virtualenv are pure Python packages it might not be necessary to load the packages used during the creation of the virtualenv, but when in doubt, it's recommended to always load the same modules used at creation time.

Installing new packages

To install packages in the virtualenv it needs to be activated, and while it is active any packages installed with pip will be actually installed inside the virtualenv itself:

Installing a Python package

[user@cluster:~]$ source venvs/venv-for-demo/bin/activate
(venv-for-demo) [user@cluster:~]$ module list
Currently Loaded Modules:
  1) gcc/7.4.0  2) python/3.7.3
(venv-for-demo) [user@cluster:~]$ pip install --no-cache-dir biopython

When installing packages it is particularly important to have the same modules that were used during the virtualenv created loaded. This is so that if any compilation is necessary during module installation the same compiler and libraries are used.

It is also important to use the --no-cache-dir option to ensure an existing pre-compiled copy of the python package is not used. Otherwise we could be mixing different compilers and architectures which could lead to hard to debug issues.

Stop using a virtualenv

To stop using a virtualenv one needs to deactivate it:
Deactivating a virtualenv

```bash
(venv-for-demo) $ deactivate
```

Removing a virtualenv

To permanently remove a virtualenv one simply deletes the directory which contains it.

Deleting a virtualenv

```bash
$ rm -rf venvs/venv-for-demo/
```

Are you looking for pip3?

pip3 is available in this way:

Loading modules to get pip3

```bash
module load gcc python
```

Check:

Show me pip3

```bash
which pip3
```

Result

```bash
/ssof/spack/humagne/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-7.4.0/python-3.7.3-51m3vikrg4nq4tjhx76tgqy7zbt4kfam/bin/pip3
```