Uni.lu HPC School 2019

PS8a: Python I (Basic)

Uni.lu High Performance Computing (HPC) Team

C. Parisot, S. Peter

University of Luxembourg (UL), Luxembourg

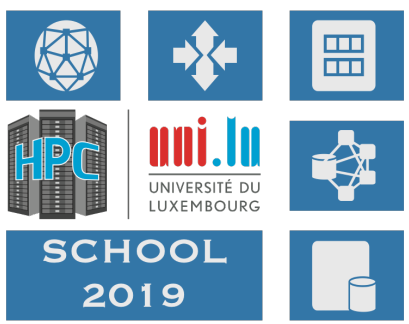
http://hpc.uni.lu

Latest versions available on Github:

UL HPC tutorials: https://github.com/ULHPC/tutorials

UL HPC School: http://hpc.uni.lu/hpc-school/

PS8a tutorial sources: ulhpc-tutorials.rtfd.io/en/latest/python/basics/



Summary

Main Objectives of this Session

Basics

- Run Python code on the cluster
- Install and use your own **Python packages**
- Create a **virtual environment**
- Compile your code in **C** to have better performances
- Use **Scoop** to distribute your code on the cluster

Advanced

- Run **Jupyter notebook** on the cluster and access to it via your browser
- Use **Celery** to distribute your code execution across the cluster

Summary

Python / Pip

- `pip`: Python package manager - "nice" python packages: `mkdocs...` - Windows: install via Chocolatey

```
$> pip install <package>                    # install <package>
```

Python / Pip

- `pip`: Python package manager - "nice" python packages: `mkdocs...` - Windows: install via Chocolatey

```
$> pip install <package>                    # install <package>
```

```
$> pip install -U pip                       # upgrade on Linux/Mac OS
```

Python / Pip

- `pip`: Python package manager - "nice" python packages: `mkdocs...` - Windows: install via Chocolatey

```
$> pip install <package>                    # install <package>
```

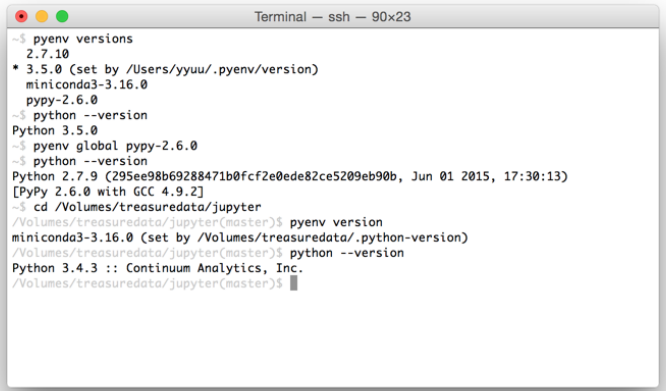
```
$> pip install -U pip                       # upgrade on Linux/Mac OS
```

- Dump python environment to a requirements file

```
$> pip freeze -l > requirements.txt         # as Ruby Gemfiles
```

Pyenv / VirtualEnv / Autoenv

- pyenv: ≃ RVM/rbenv for Python
- virtualenv ≃ RVM Gemset
- (optional) direnv - Directory-based shell environments - Rely on direnv to load pyenv and virtualenv. Just write a `.envrc` in your project directory **Ex**:




```
# (rootdir)/.envrc : direnv configuration file
pyversion=$(head .python-version)
pvenv=$(head     .python-virtualenv)
use python ${pyversion}
# Create the virtualenv if not yet done
layout virtualenv ${pyversion} ${pvenv}
# activate it
layout activate ${pvenv}-${pyversion}
```

Summary

Python on UL HPC Platform

You have 2 different way of using Python on UL HPC Platform

- Use the **system** Python installed on the node
  - ↪ only version **2.7** is installed under `/usr/bin/python`
- Rely on Environment Modules **once** on a computing node
  - ↪ then you can search for version of Python available: `module avail lang/python`

Python on UL HPC Platform

You have 2 different way of using Python on UL HPC Platform

- Use the **system** Python installed on the node
  - ↪ only version **2.7** is installed under `/usr/bin/python`
- Rely on Environment Modules **once** on a computing node
  - ↪ then you can search for version of Python available: `module avail lang/python`

```
---------------- /opt/apps/resif/data/stable/default/modules/all ----------
   lang/Python/2.7.14-foss-2018a-bare
   lang/Python/2.7.14-foss-2018a
   lang/Python/2.7.14-GCCcore-6.4.0-bare
   lang/Python/2.7.14-intel-2018a-bare
   lang/Python/2.7.14-intel-2018a
   lang/Python/3.6.4-foss-2018a-bare
   lang/Python/3.6.4-foss-2018a
   lang/Python/3.6.4-intel-2018a-bare
   lang/Python/3.6.4-intel-2018a        (D)
```

- The first thing you should do is always to load the version of Python that you need
  - ↪ Your scripts will use the module version load to execute
  - ↪ This version will be used inside your virtual environment
- **bare** versions doesn't include any packages!
  - ↪ **non bare** version provides 43 Python packages that can be useful for you
  - ↪ **bare** version contains only **pip** and **setuptools**, you have to install every package yourself

Examples of module usage

```
$(node)> module load lang/Python/3.6.4-foss-2018a
$(node)> python --version
Python 3.6.4

$(node)> module load lang/Python/2.7.14-foss-2018a
$(node)> python --version
Python 2.7.14

$(node)> module purge
$(node)> python --version
Python 2.7.5
```

Summary

Virtualenv

Virtualenv allows you to create an environment for Python and isolate the installation of the packages inside it.
You can have several virtual environment that are independant to each other and you can load them to use the packages installed inside.

- For bare version of Python, you will have to **install** virtualenv package locally in your home directory by using **pip**

```
$(node)> module load lang/Python/3.6.4-foss-2018a-bare
$(node)> python -m pip install --no-cache-dir --user virtualenv
Installing collected packages: virtualenv
Successfully installed virtualenv-16.6.1
$(node)> python -m virtualenv --version
16.6.1
```

- If you use **non-bare** version of Python, it comes with a `virtualenv` already installed

```
$(node)> module load lang/Python/2.7.14-foss-2018a
$(node)> python -s -m virtualenv --version
15.1.0
$(node)> module load lang/Python/3.6.4-foss-2018a
$(node)> python -s -m virtualenv --version
15.1.0
```

- Create your own **virtual environment** to install packages in it

```
$(node)> module load <your Python version>
$(node)> python -m virtualenv <name of your environment>
```

- If you don't load any module, the the **system** python will be used. It is a best practice to specify your Python version via module before running any script.
- Now you can **activate** the environment named `venv` with the following command

```
$(node)> source venv/bin/activate
(venv) $(node)> # you are now inside your virtual environment
```

Summary

Pythran

- Pythran is an ahead of time compiler for a subset of the Python language, with a focus on scientific computing.
- compile your Python code in C++ for (hopefully) faster execution

Summary

SCOOP

- parallelisation using Scoop
- SCOOP (Scalable COncurrent Operations in Python)
- distributed task module allowing **concurrent parallel programming** on various environments, from heterogeneous grids to supercomputers
- SCOOP features and advantages:
  - ↪ Harness the power of multiple computers over network;
  - ↪ Ability to spawn multiple tasks inside a task;
  - ↪ API compatible with PEP-3148;
  - ↪ Parallelizing serial code with only minor modifications;
  - ↪ Efficient load-balancing.

Summary

Jupyter Notebook

- extends the console-based approach to **interactive computing**
- provides web-based application for capturing the whole computation process: **developing**, **documenting**, and **executing code**, as well as **communicating the results**
- It combines two components:
  - ↪ **A web application**: a browser-based tool for interactive authoring of documents
  - ↪ **Notebook documents**: a representation of all content visible in the web application
- Can be used with different langagues (Python, R, . . .)

Summary

Celery - Distributed Task Queue

- Celery
- simple, flexible, and reliable **distributed system** to process **vast amounts of messages**
- **task queue** with focus on real-time processing, while also supporting **task scheduling**.
- large and diverse community of users and contributors
- can use **Slurm** and **MPI** to distribute the tasks to the workers

Questions?

http://hpc.uni.lu

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement Parisot
Dr. Fréderic Pinel
Dr. Emmanuel Kieffer

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu