UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

Software installation on BioHPC

[email] biohpc-help@utsouthwestern.edu
[web] portal.biohpc.swmed.edu

Topics covered

Basic principles of (Linux) software

- *The $PATH variable*
- *Scripted vs compiled programs*

Python software

- *pip*
- *virtual environments*
- *Anaconda*

Generic software

- *Installation from source code*
  ~~*Installation with EasyBuild*~~

Pre-requisites

To make the most of this tutorial, you should already know how to:

- *Login with SSH to a linux machine*
- *Navigate directories in linux terminal*
- *Edit a textfile in terminal (eg nano, vi etc)*

Basic principles of (Linux) software

The $PATH variable

In Linux (and other OS also) the $PATH is a global variable that contains a list of locations:

```
[s200969@Nucleus005 ~]$ echo $PATH
/cm/shared/apps/slurm/16.05.8/sbin:/cm/shared/apps/slurm/16.05.8/bin:/usr/local/bin
:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin:/sbin:/usr/sbin:/cm/local/apps
/environment-modules/3.2.10/bin:/home2/s200969/bin
```

The CLI will consider these locations when interpreting a command name as a program.

```
[s200969@Nucleus005 ~]$ which python
/usr/bin/python
[s200969@Nucleus005 ~]$ module load python/3.3.2
[s200969@Nucleus005 ~]$ which python3
/cm/shared/apps/python/3.3.2/bin/python3
```

Basic principles of (Linux) software

Modifying $PATH variable

Let assume there is a program called repeat located in a personal folder:

```
[s200969@Nucleus005 software]$ pwd
/project/biohpcadmin/s200969/software
[s200969@Nucleus005 software]$ ls
repeat
[s200969@Nucleus005 software]$ ./repeat One Two Three
One Two Three
One Two Three
```

However, this program only works while in that folder or with an absolute path:

```
[s200969@Nucleus005 software]$ cd ..
[s200969@Nucleus005 s200969]$ ./repeat One Two Three
-bash: ./repeat: No such file or directory
[s200969@Nucleus005 s200969]$ /project/biohpcadmin/s200969/software/repeat One Two Three
One Two Three
One Two Three
```

Adding the program location to the $PATH variable, makes the command work everywhere:

```
[s200969@Nucleus005 ~]$ export PATH=$PATH:/project/biohpcadmin/s200969/software
[s200969@Nucleus005 ~]$ repeat One Two Three
One Two Three
One Two Three
```

Basic principles of (Linux) software

Permanently setting $PATH variable

The command 'export $PATH' will be useful only for the current session and only for the current Nucleus node.
If you log out, or log-in somewhere else, the $PATH will not contain the changes we made.

In order to make the $PATH change permanent, we need to edit the file **~/.bash_profile**

```
[s200969@Nucleus005 ~]$ cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:/project/biohpcadmin/s200969/software

export PATH
```

Be careful! Only alter your $PATH once you are certain you need to.
If the same program exists in multiple locations of $PATH, then the first one takes precedence.

Basic principles of (Linux) software

Scripted vs compiled programs

Scripted programs	Compiled programs

- The program is a script (a text file of commands in order)
 	
- The script is executed by an interpreter (eg python), which runs the program line by line as scripted
 	
- You only need the script file and the interpreter to run your program
 	
BioHPC has many interpreters already available.
	- The script/code does not run directly as it is but needs to be compiled and built using an appropriate tool
	- This will result in a program being created from the original source
	- Compiling a source code requires a specific compiler:
	  - To match the language of the source
	  - To match the architecture of the target environment

- Slow execution - Easy to debug errors	- Fast execution - Hard to debug errors

Python software

Python is a script interpreter. Python programs are scripted programs.

Generally, complex programs will have more than 1 single script file. The term **package** or **module** is often used in python. A package or module is a collection of script files necessary to make up a complex program.

Installing a python package, means obtaining the package and placing it in a specific location already known to the python interpreter.
Similar to the linux $PATH variable, python has its own path called **`sys.path`** where it will look for packages.

```
>>> import sys
>>> print '\n'.join(sys.path)
/usr/lib64/python27.zip
/usr/lib64/python2.7
/usr/lib64/python2.7/plat-linux2
/usr/lib64/python2.7/lib-tk
/usr/lib64/python2.7/lib-old
/usr/lib64/python2.7/lib-dynload
/home2/s200969/.local/lib/python2.7/site-packages
```

Python software

pip

In python we never change the `sys.path` manually. Instead, we use package manager to receive the source package and place it in the appropriate.
PIP is the foremost python package manager. Almost all published python packages can be fetched with the command

```
pip install <package-name>
```

Can you think of any issues that may arise while installing packages this way?

Python software

pip

In python we never change the `sys.path` manually. Instead, we use package manager to receive the source package and place it in the appropriate.
PIP is the foremost python package manager. Almost all published python packages can be fetched with the command

```
pip install <package-name>
```

Can you think of any issues that may arise while installing packages this way?

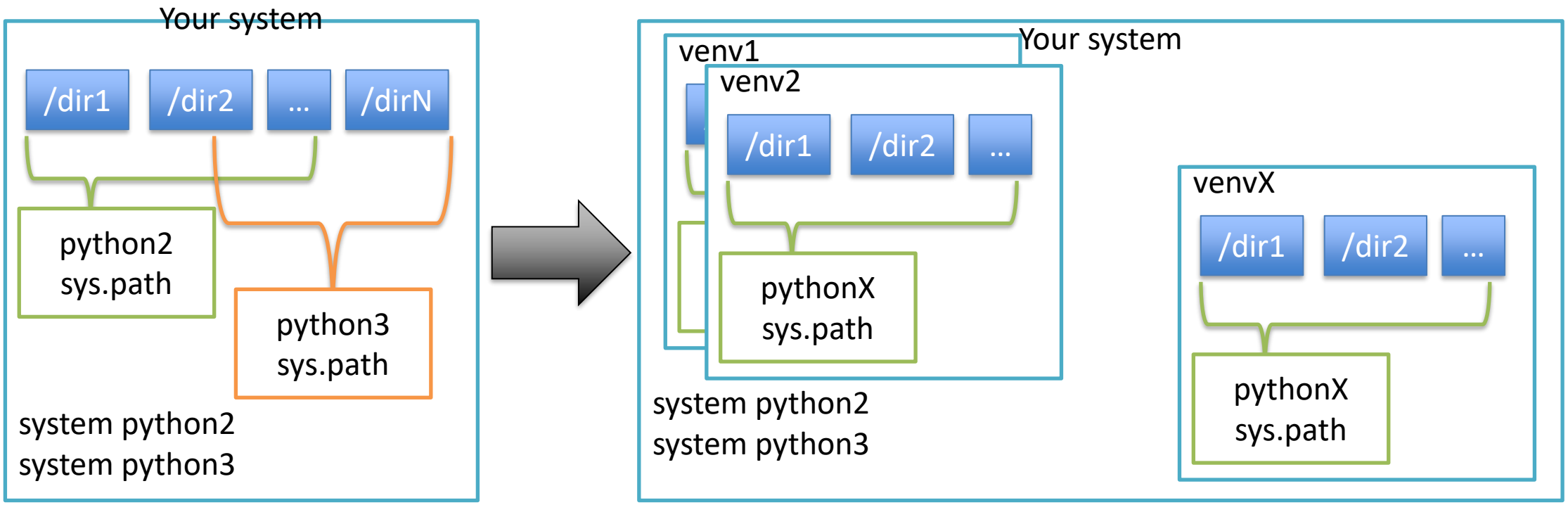
- Where is the package installed?
- What if you need several versions of the same package?
  e.g. packageX==2.0 works with python2 and packageX==2.1 works with python3
- What happens in the long run if you 'install and forget'?
- It can affect your system-wide experience!

Don't panic! There are plenty of solutions to this conundrum.

Python software

Virtual environments

Python Virtual Environments (**venv**) are a way to encapsulate a certain python version + a collection of certain packages. This allows you to create environments for each project (or group of related projects) in order to ensure you will always have the necessary packages and python version for that project.



Python software

Virtual environments

Create a virtual environment

```
[s200969@Nucleus006 ~]$ cd /project/biohpcadmin/s200969/software/
[s200969@Nucleus006 software]$ mkdir venv_example
[s200969@Nucleus006 software]$ module load python/3.6.4-anaconda
[s200969@Nucleus006 software]$ pip install --user virtualenv
[s200969@Nucleus006 software]$ python -m virtualenv venv_example
created virtual environment CPython3.6.4.final.0-64 in 2279ms
```

Activate environment / Use / Deactivate

```
[s200969@Nucleus006 software]$ source venv_example/bin/activate
(venv_example) [s200969@Nucleus006 software]$ pip install cowsay
Successfully installed cowsay-4.0
(venv_example) [s200969@Nucleus006 software]$ cowsay hello
(venv_example) [s200969@Nucleus006 software]$ deactivate
[s200969@Nucleus006 software]$ cowsay hello
bash: cowsay: command not found...
```

Python software

So far:

pip: the package manager
virtualenv: the environment manager, installable via pip

This approach is long proved to work and hardened by experience.
More often than not it is enough to encapsulate projects of any size.

However; there are downfalls:
- Cross dependency management is still hard for larger (many packages) projects

For these scenarios, solutions exist!

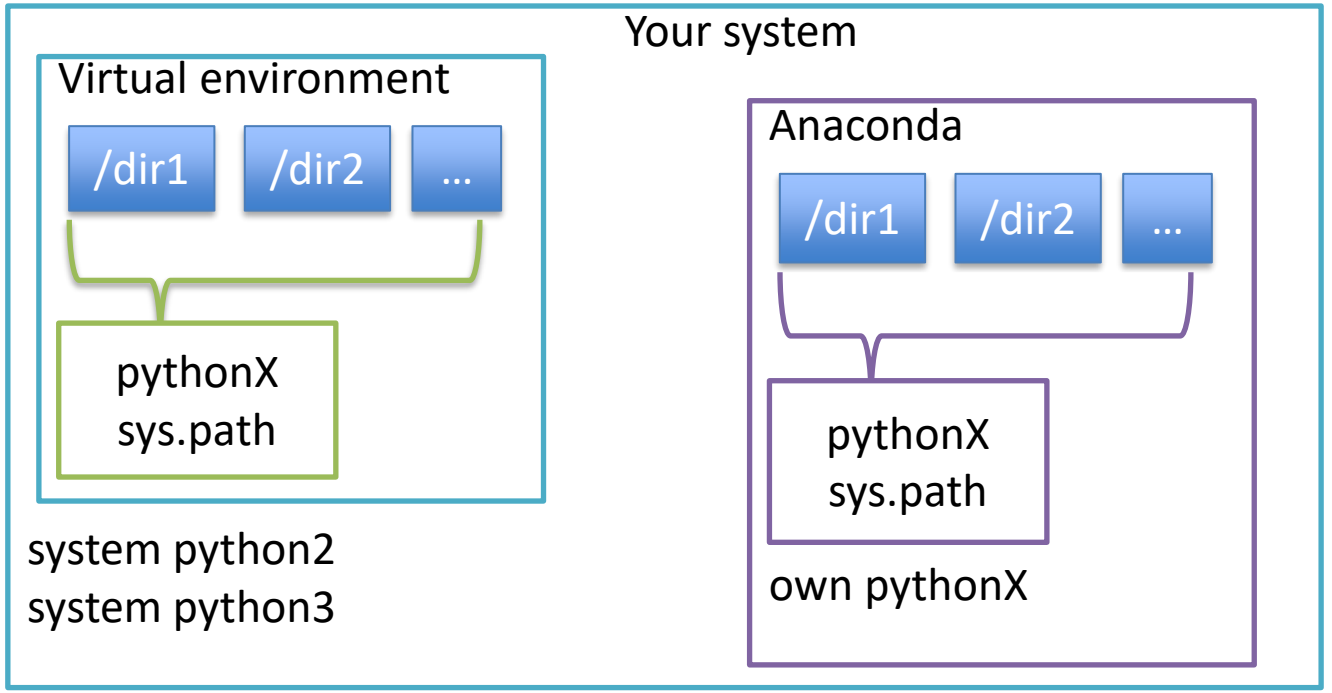
Python software

Anaconda

Anaconda is a Platform, it contains many utilities and features.
The purpose of anaconda, is to encapsulate entire systems of applications.
Anaconda works in python, but also supports other languages.

In fact, Anaconda is similar to having a special virtualenv which contains also python itself.

Anaconda also uses environments, to the same logic as virtualenv.
Anaconda environments are superior to virtual env because of better dependency management and multiple language support.

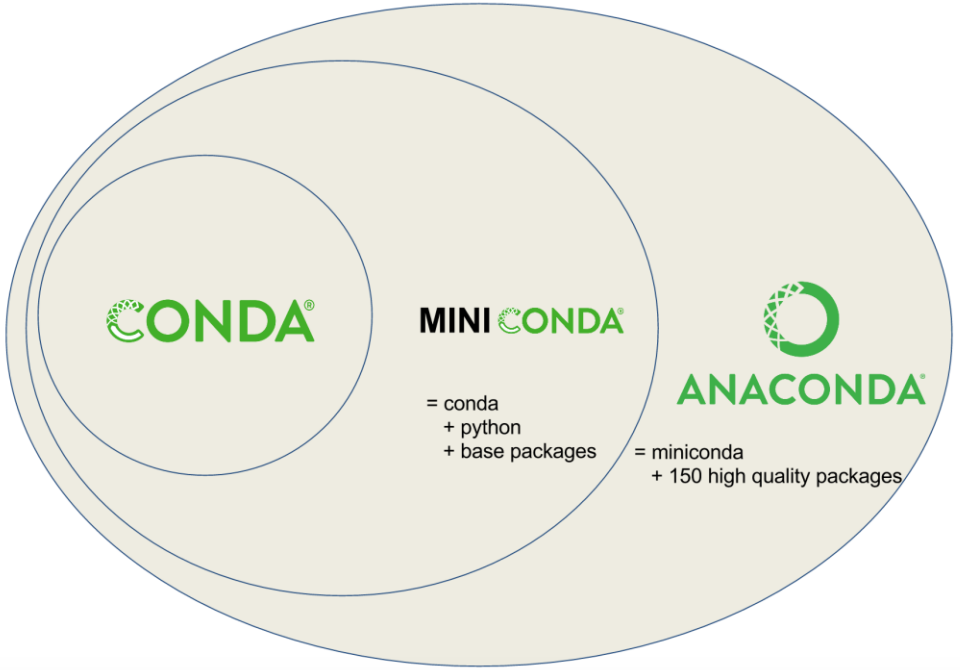


Python software

Conda, MiniConda, Anaconda
Install miniconda:

```
curl -LO
https://repo.anaconda.com/miniconda/Miniconda3-
latest-Linux-x86_64.sh

bash Miniconda3-latest-Linux-x86_64.sh
```



Use conda environments similar to virtual environments:

```
conda create env-name
conda activate env-name
conda install (instead of pip install)
```

Nice intro here:
https://astrobiomike.github.io/unix/conda-intro

Generic software

Installation from source code

- You need to obtain the source code of the application
  - Generally through a git url
  - Sometimes as a compressed archive (in this case, you need to un-compress it)

- Expect high quality software to have its own documentation regarding installation.
- More often than not the code contains a Makefile
  - specifies the steps and architecture, you only need to ‘make’ the makefile

Your take home message for BioHPC systems:

Generally, you will get a permission error. This doesn’t mean that you don’t have permission to install software, just that the generic installation steps might try to write to a path that you don’t own.
Explore the documentation for custom installations, in order to place the install in a folder where you have access to (eg /project)
Generally the keywords for this are ***prefix*** or ***install-dir***.

Generic software

Installation from source code

https://github.com/cowsay-org/cowsay

```
[s200969@Nucleus006 software]$ git clone https://github.com/cowsay-org/cowsay.git
Cloning into 'cowsay'...
s200969@Nucleus006 software]$ cd cowsay/
[s200969@Nucleus006 cowsay]$ make install  prefix=/project/biohpcadmin/s200969/software/my-cowsay-
installation

[s200969@Nucleus006 cowsay]$ cd ../my-cowsay-installation/

[s200969@Nucleus006 my-cowsay-installation]$ ./bin/cowsay hello


Add to $PATH if necessary
```

Thank you for your attention

Regarding BioHPC policy:

- You are responsible for the software you install
  - consider quality and trustworthiness of the software you chose
- You may only install to your accessible locations

- Cluster-wide installation is still possible with request to BioHPC Helpdesk (biohpc-help@utsouthwestern.edu)

Questions / Comments / Remarks ?