UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

Software Installation on BioHPC

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

Updated for 2020-07-15

Software installation - Windows

Topics covered

Basic principles of (Linux) software

The $PATH variable
Scripted vs compiled programs

Python software

pip
virtual environments
Anaconda

R package installation

Troubleshooting

Generic software

Installation from source code

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 text file in terminal (e.g. 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:

```
[s201048@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/loca
l/apps/environment-modules/3.2.10/bin:/home2/s201048/.local/bin:/work/biohpcadmi
n/s201048/petsc_2/valgrind-3.17.0/bin
```

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

```
[s201048@Nucleus005 ~]$ which python
/usr/bin/python
[s201048@Nucleus005 ~]$ module add python/3.8.x-anaconda
[s201048@Nucleus005 ~]$ which python3
/cm/shared/apps/python/3.8.x-anaconda/bin/python3
```

Basic principles of (Linux) software

Modifying the $PATH variable

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

```
[s201048@Nucleus005 bin]$ pwd
/home2/s201048/bin
[s201048@Nucleus005 bin]$ ls
hello  hello.c
[s201048@Nucleus005 bin]$ ./hello
Hello World
[s201048@Nucleus005 bin]$ cd ../
[s201048@Nucleus005 ~]$ ./hello
-bash: ./hello: No such file or directory
[s201048@Nucleus005 ~]$ /home2/s201048/bin/hello
Hello World
```

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

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

```
[s201048@Nucleus005 ~]$ export PATH=$PATH:/home2/s201048/bin
[s201048@Nucleus005 ~]$ hello
Hello World
```

Basic principles of (Linux) software

Permanently setting the $PATH variable

The command 'export $PATH' will be useful only for the current session and only for the current Nucleus node. If you logout, 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**

```
[s201048@Nucleus005 ~]$ cat ~/.bash_profile
# .bash_profile

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

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

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.

Other PATH like variables

- LD_LIBRARY_PATH
  - is the search path environment variable for the linux shared library
- PYTHONPATH
  - is an environment variable which you can set to add additional directories where python will look for modules and packages.
  - For most installations, you should not set these variables since they are not needed for Python to run. Python knows where to find its standard library.
  - The only reason to set PYTHONPATH is to maintain directories of custom Python libraries that you do not want to install in the global default location
- LIBPATH
  - is for the compiler, helps it find metadata files. Like type libraries, .NET assemblies, WinRT .winmd files.

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 (e.g. 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 a 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/s201048/.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?

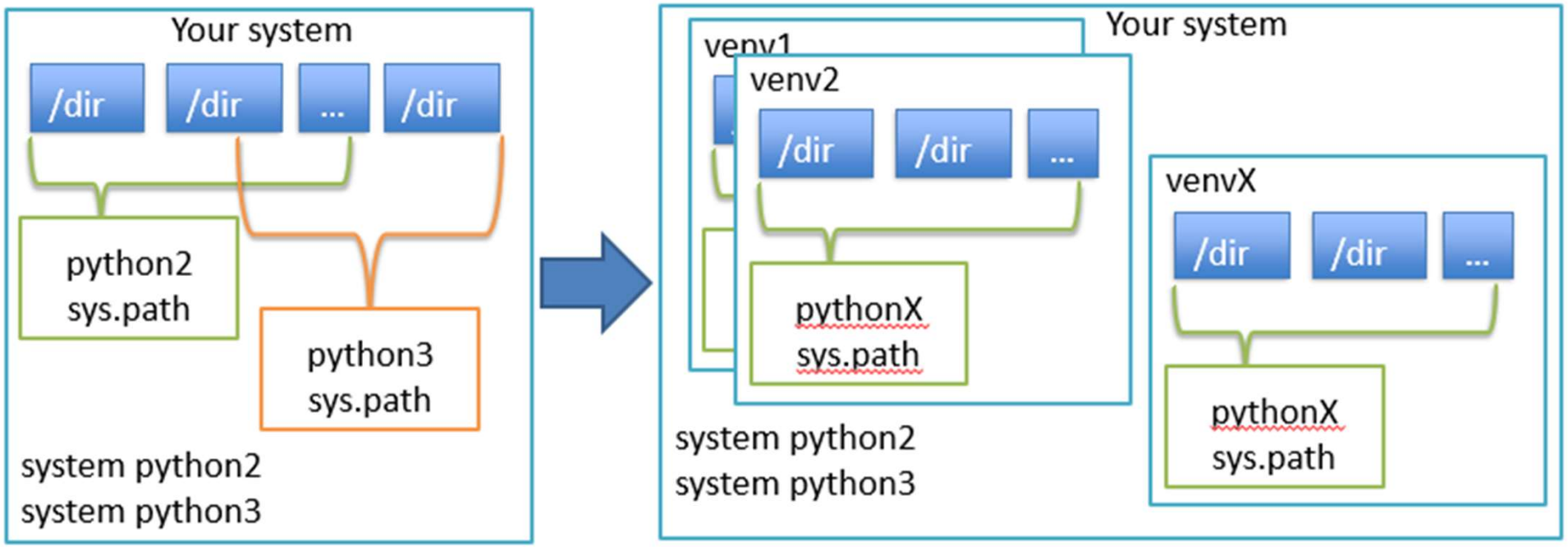
- Where is the package installed?
- What if you need several versions of the same package?
  e.g. packageX==2.0 works with python2 and packageX==2.1 works with python3
- What happens in the long run if you 'install and forget'?
- It can affect your system-wide experience!

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

Install virtualenv package

```
[s201048@Nucleus005 ~]$ cd /work/biohpcadmin/s201048/software
[s201048@Nucleus005 software]$ mkdir venv_example
[s201048@Nucleus005 software]$ module add python/3.8.x-anaconda
[s201048@Nucleus005 software]$ pip install --user virtualenv
```

Create a virtual environment

```
[s201048@Nucleus005 software]$ python -m virtualenv venv_example
created virtual environment CPython3.8.8.final.0-64 in 7861ms
```

Activate environment / Use / Deactivate

```
[s201048@Nucleus005 software]$ source venv_example/bin/activate
(venv_example) [s201048@Nucleus005 software]$ pip install cowsay
Successfully installed cowsay-4.0
(venv_example) [s201048@Nucleus005 software]$ cowsay hello
(venv_example) [s201048@Nucleus005 software]$ deactivate
[s201048@Nucleus005 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.
Often it is enough to encapsulate projects of any size.

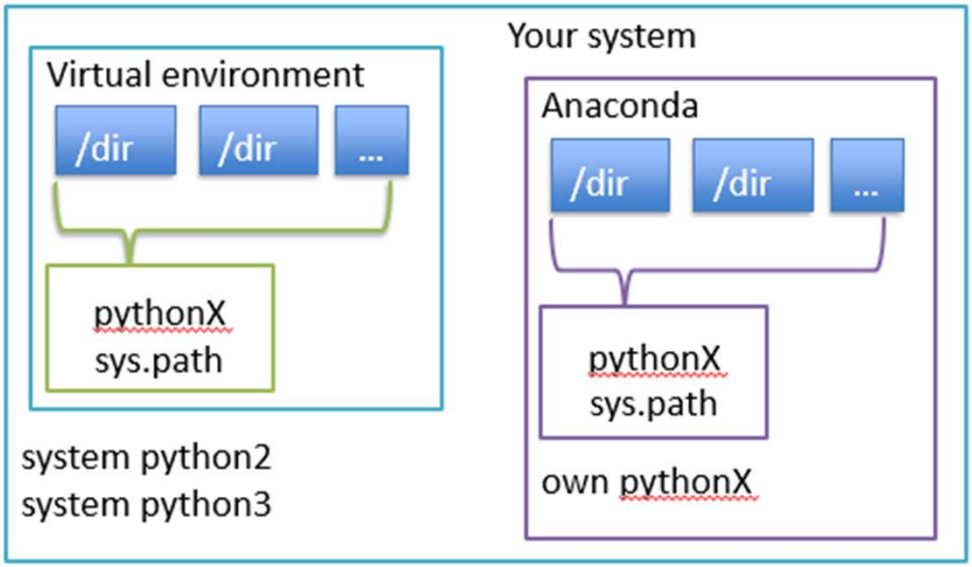
However; there are downfalls:
- Cross dependency management is still hard for larger (many packages) projects

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 virtualenv because of better dependency management and multiple language support.

Python Software

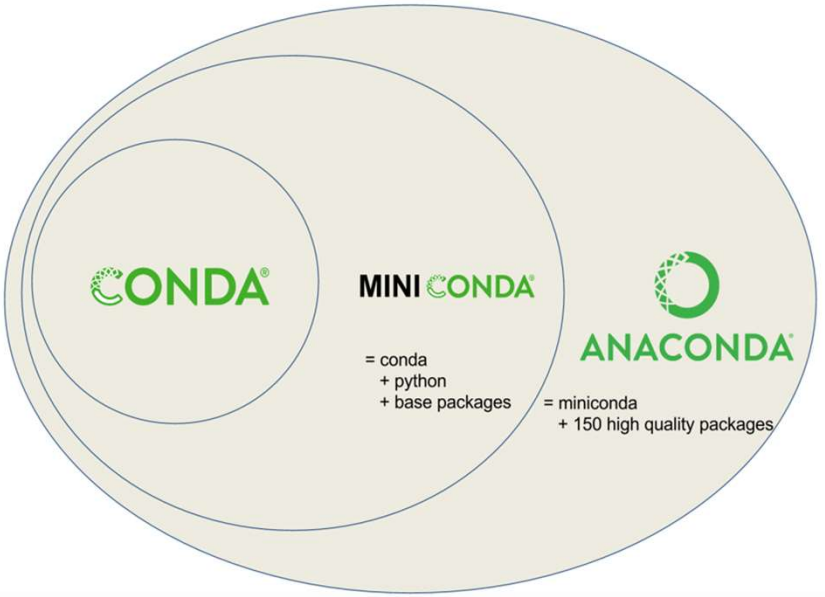
Conda, Miniconda, Anaconda:

```
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

R package installation

- Install packages using *RStudio GUI* or *install.packages("cowsay")*
- Where are your R packages installed?

  > .libPaths()

  [1] "/home2/***<username>***/R/x86_64-pc-linux-gnu-library/***3.6***" #your own R packages

  [2] "/cm/shared/apps/R/gcc/**3.6.1**/lib64" #system packages

  [3] "/cm/shared/apps/R/gcc/**3.6.1**/lib64/R/library" #system packages

  `Error installing R package?`

- Rstudio server or Rstudio OnDemand has error when installing a package:
  - Use corresponding R version from the terminal to install the package and use it in Rstudio server (3.3.2) or Rstudio OnDemnd, such as 3.6.1, they share the same library.

Missing dependency for R package

```
libgfortran.so.4: cannot open shared object file:
No such file or directory
```

- Check the modules, often the dependency is installed as a module, using:

  module avail <package name>

- If not found, send BioHPC a ticket. We may need to install that package for you:

  yum install <package name>

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.
- Often 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***.

Steps to install a generic software

```
#Basic example
./configure
make
Make install

#More realistic example
./configure --prefix=/home2/s173217/.local
make –j <n number>
Make install
```

- Ref to https://thoughtbot.com/blog/the-magic-behind-configure-make-make-install

Generic software

Installation from source code:

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

```
[s201048@Nucleus006 software]$ git clone https://github.com/cowsay-org/cowsay.git
Cloning into 'cowsay'...
[s201048@Nucleus006 software]$ cd cowsay/
[s201048@Nucleus006 cowsay]$ make install
prefix=/work/biohpcadmin/s201048/software/my-cowsay-installation

[s201048@Nucleus006 cowsay]$ cd ../my-cowsay-installation/

[s201048@Nucleus006 my-cowsay-installation]$ ./bin/cowsay hello

Add to $PATH if necessary
```

Demo

Setup the node and modules

```
srun -p GPUA100 --pty /bin/bash
module load python/3.7.x-anaconda
module load cuda112/toolkit/11.2.0
module add cudnn/8.1.1.33
```

Virtual environment

```
mkdir /path/t        /virtual/env
cd /path/t        /new/v      l/env
python -      /ve       env
source v      /bin/      va
pip instal      sorflo      ==1.14.0
Pip install ke          .4
```

Anaconda environment

```
conda create -n student -y
conda activate student
conda install -c anaconda tensorflow-gpu=1.14.0 -y
conda install -c anaconda keras==2.2.4 -y
```

Evaluate the environment

```
python
>>> import tensorflow as tf
>>> print(tf.__version__)
1.14.0
>>> import keras
Using TensorFlow backend.
```

Proxy setup on compute nodes

Can not connect type of errors:

Anaconda

- cat $USER/.condarc
- proxy_servers:
  http: http://proxy.swmed.edu:3128
  https: http://proxy.swmed.edu:3128

R

- Sys.setenv(https_proxy = 'http://proxy.swmed.edu:3128')

Linux terminal

- cat ~/.bashrc
- export http_proxy=http://proxy.swmed.edu:3128
  export https_proxy=http://proxy.swmed.edu:3128

Thank you

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 ?