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 2023-09-13

Topics

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

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 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. 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.

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.
- 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

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.

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.

```
>>> 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

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

PIP is the foremost python package manager. Almost all published python packages can be fetched with the command
pip install <package-name> Or pip install –r requirements.txt

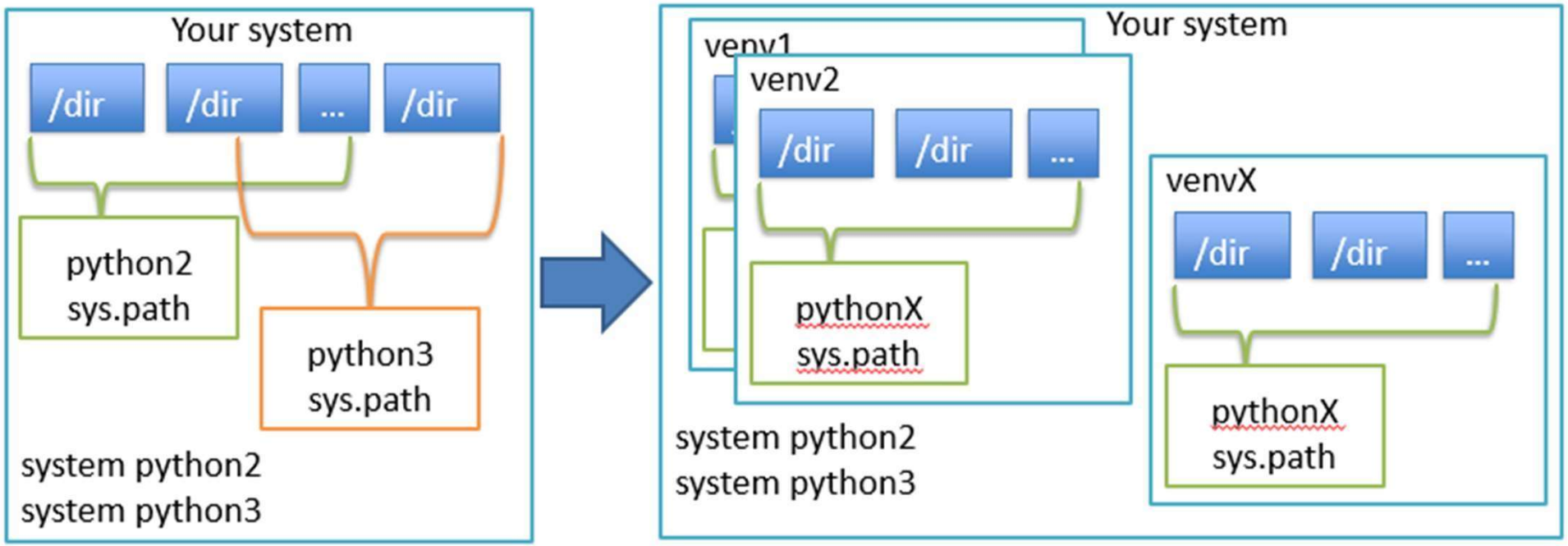
Possible issues that may arise?

- Where is the package installed?
- What happens in the long run if you 'install and forget'?
- 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

Fortunately, there are plenty of solutions to this conundrum.

Python Virtual Environment (venv)

It is 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.



How to use venv

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...
```

Pros and Cons

- Virtualenv is the most common and easy to install tool for virtual environments. It's a great tool for beginners.
- It has lots of documentation for many issues.

However; there are downfalls:

- Cross dependency management is still hard for larger (many packages) projects
  - Needs different versions of python interpreters
  - where packages dependency conflict happens and not easy to manually fix.

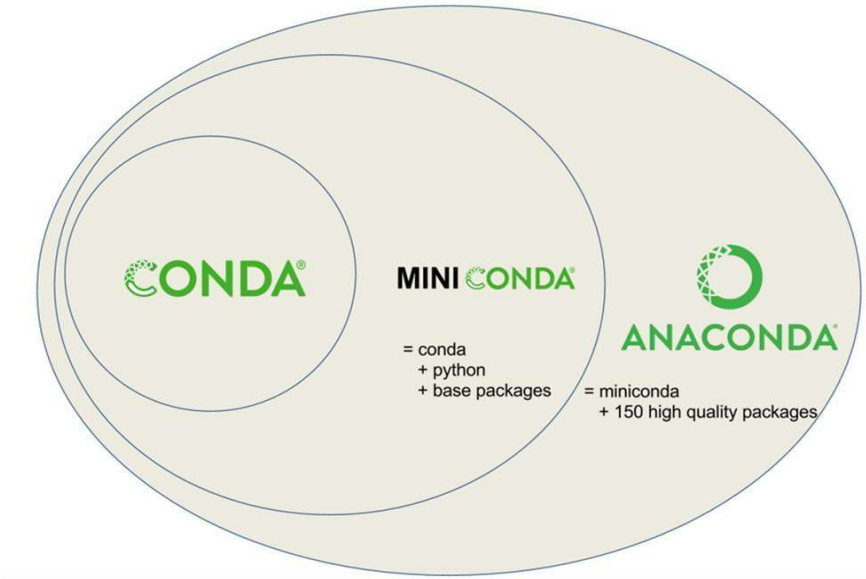
```
spyder 4.2.5 requires jedi==0.17.2, but you have jedi 0.18.2 which is incompatible.
spyder 4.2.5 requires parso==0.7.0, but you have parso 0.8.3 which is incompatible.
```

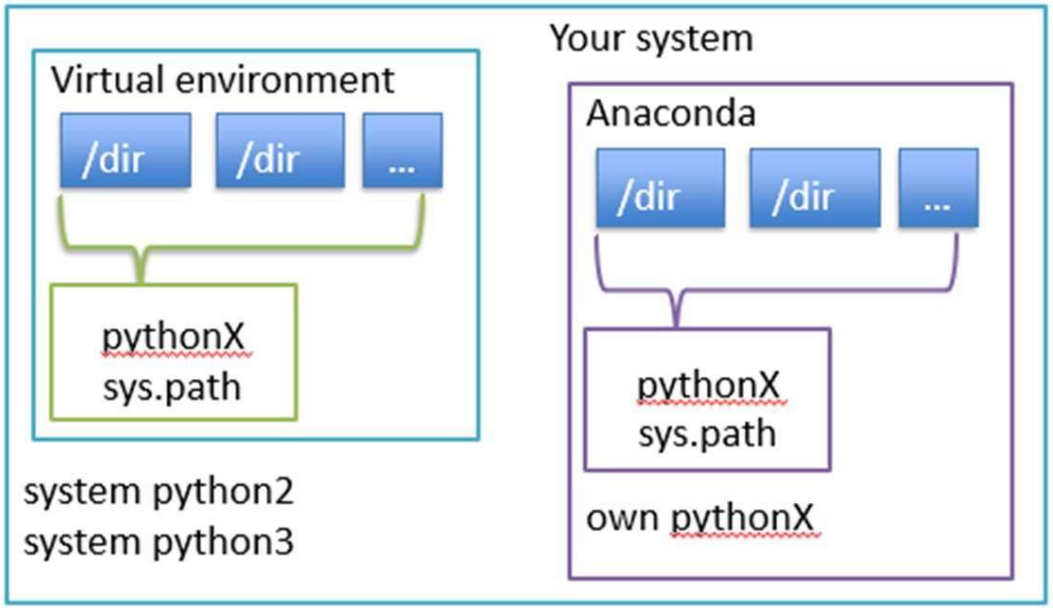
For these scenarios, solutions exist!

Python Software - Anaconda

Similar to the python venv, anaconda is also able to encapsulate entire systems of applications, but it can do something more.

- Support packages written by other languages, e.g. R, C/C++
- Can work with Jupyter notebook together by creating a kernel
- Easy to reproduce the environment





Python Software - Anaconda

Miniconda installation – Do not need to do this in Nucleus cluster

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

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

We have already installed anaconda in Nucleus cluster

```
[s216882@Nucleus006 ~]$ module av python

--------------------------- /cm/shared/modulefiles ---------------
python/2.7.14-anaconda              python/3.4.x-anaconda
python/2.7.3-epd                    python/3.6.1-2-anaconda
python/2.7.5                        python/3.6.4-anaconda
python/2.7.6-epd                    python/3.7.x-anaconda
python/2.7.x-anaconda               python/3.8.x-anaconda
python/3.10.x-anaconda              python/latest-3.11.x-anaconda
python/3.11.x-anaconda              python-matlab-api/2019b/python36
python/3.3.2
```

Conda install

Google key words “conda install package-name”

Common commands

```
module load python/3.10.x-anaconda
```

```
conda create -n my-env
```
or
```
conda create -p /path/to/my-env
```

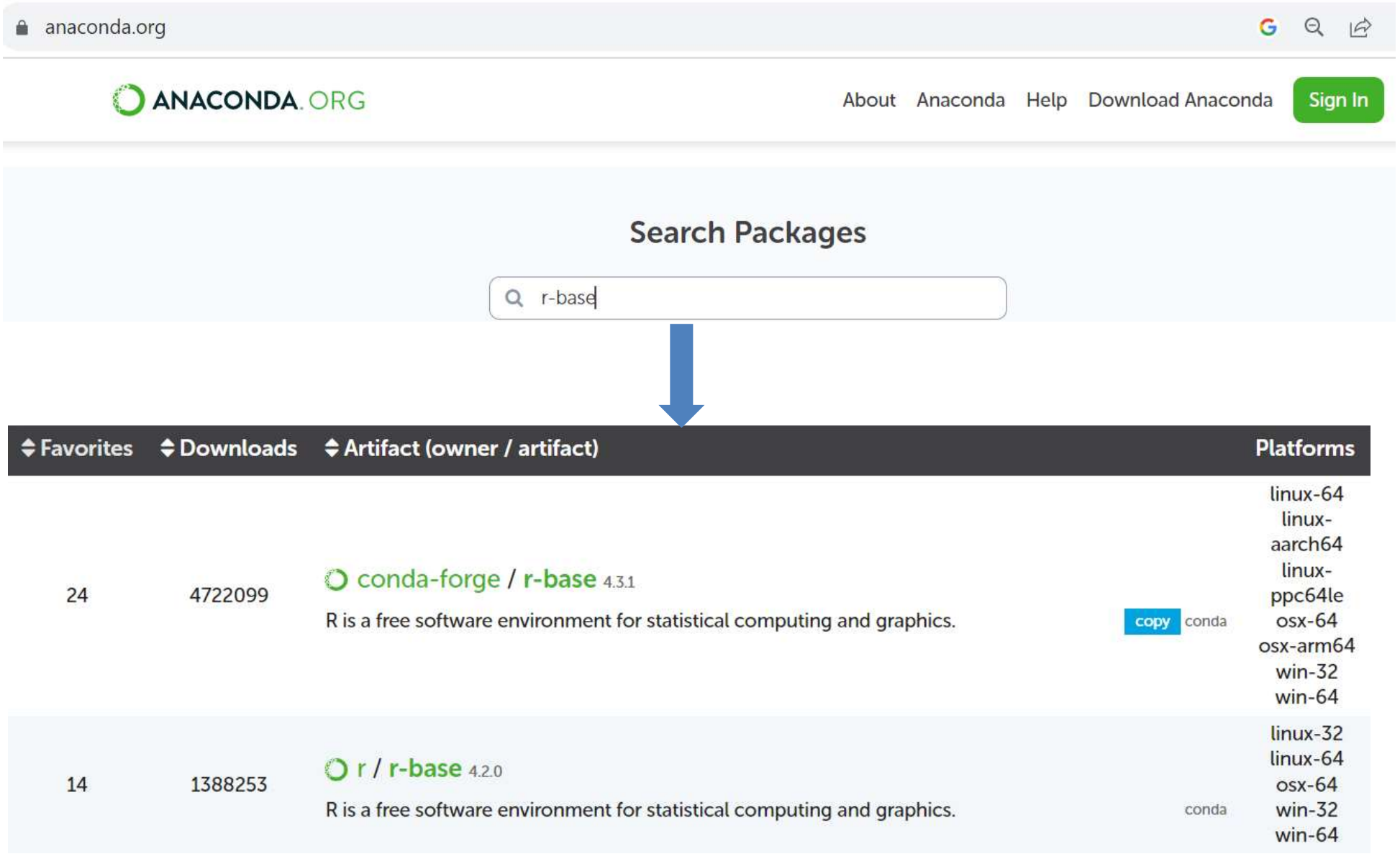
```
conda activate my-env
```
or
```
conda activate /path/to/my-env
```

```
conda install -c conda-forge tmux
```

```
conda deactivate
```

Note: conda will install everything under your home directory by default, unless you indicate the specific path using –p option

Search available packages



Search in terminal

```
(test) [s216882@Nucleus006 ~]$ conda search -f star
Loading channels: done
# Name                       Version           Build  Channel
star                          2.4.0j               0  bioconda
star                          2.4.0j               1  bioconda
star                          2.4.0j       h9ee0642_2  bioconda
star                          2.4.2a               0  bioconda
star                          2.5.0a               0  bioconda
star                          2.5.0b               0  bioconda
star                          2.5.0c               0  bioconda
star                          2.5.1b               0  bioconda
star                          2.5.2a               0  bioconda
star                          2.5.2b               0  bioconda
star                          2.5.3a               0  bioconda
star                          2.5.4a               0  bioconda
star                          2.6.0b               0  bioconda
```

```
(test) [s216882@Nucleus006 ~]$ conda install -c bioconda star=2.7.10b
Collecting package metadata (current_repodata.json): done
Solving environment: \
```

Export your environment

You can share all the components in any env by export it into a file using **conda env export –f my-env.yml**

Your collaborator can reproduce this environment by **conda env create –p <path/to/install> -f my-env.yml**

```
(test) [s216882@Nucleus006 ~]$ conda env export
name: test
channels:
  - bioconda
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=conda_forge
  - _openmp_mutex=4.5=2_gnu
  - bzip2=1.0.8=h7f98852_4
  - c-ares=1.19.1=hd590300_0
  - ca-certificates=2023.08.22=h06a4308_0
  - htslib=1.18=h81da01d_0
  - keyutils=1.6.1=h166bdaf_0
  - krb5=1.21.2=h659d440_0
  - libcurl=8.2.1=hca28451_0
  - libdeflate=1.18=h0b41bf4_0
  - libedit=3.1.20221030=h5eee18b_0
  - libev=4.33=h516909a_1
  - libgcc-ng=13.2.0=h807b86a_0
  - libgomp=13.2.0=h807b86a_0
  - libnghttp2=1.52.0=h61bc06f_0
  - libssh2=1.11.0=h0841786_0
  - libstdcxx-ng=13.2.0=h7e041cc_0
  - libzlib=1.2.13=hd590300_5
  - ncurses=6.4=hcb278e6_0
  - openssl=3.1.2=hd590300_0
  - star=2.7.10b=h6b7c446_1
  - xz=5.4.2=h5eee18b_0
  - zlib=1.2.13=hd590300_5
  - zstd=1.5.5=hfc55251_0
prefix: /home2/s216882/.conda/envs/test
```

Create a python kernel

$ ssh yourusername@nucleus.biohpc.swmed.edu

$ module load python/3.7.x-anaconda

$ conda create --name foo python=3.7

$ conda activate foo

(foo) $ pip install ipykernel

(foo) $ pip install *<additional modules as needed>*

(foo) $ python -m ipykernel install --user --name foo --display-name "my foo env“

(foo) $ conda deactivate

1) create conda env
2) install packages
3) create jupyter kernel
4) exit conda env

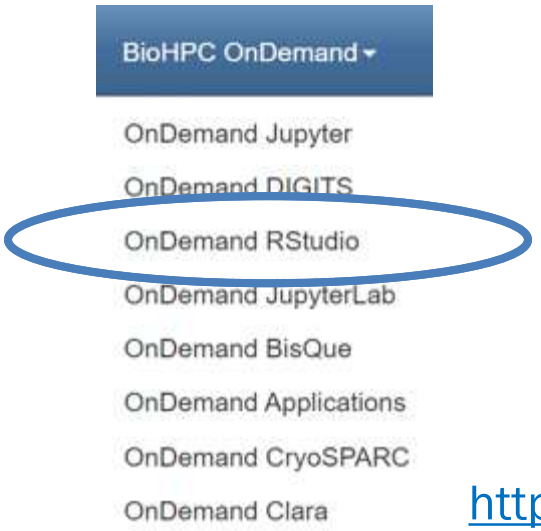
Then you can create and use this foo kernel in BioHPC Ondemand Jupyter.
Click this Link to Jupyter Notebook training slides

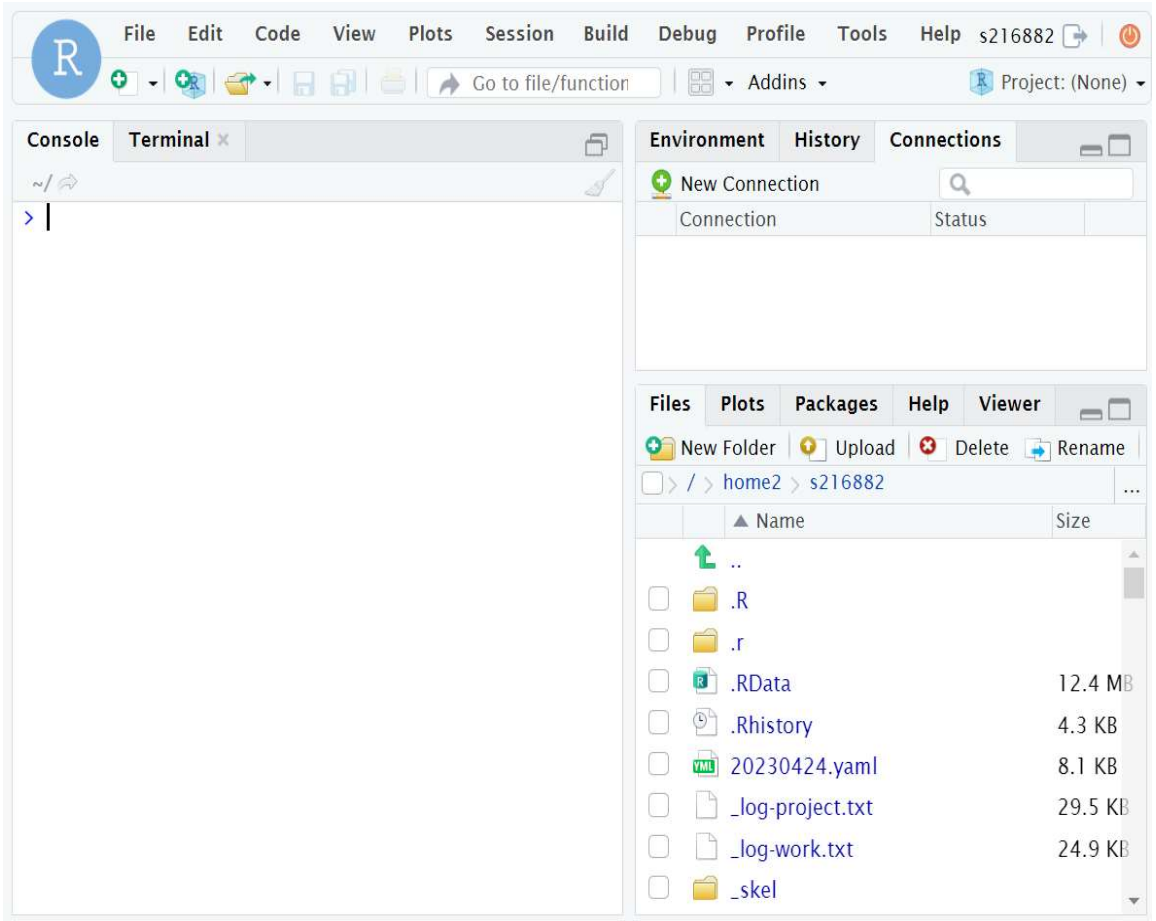
Intro to Ondemand Rstudio

Standard 20 hr limit

Whole node to yourself

You can choose R/3.3.2, 3.4.1, 3.5.1 with Seurat, R/3.6.1, R/4.0.2, R/4.1.1, R/4.2.2





https://portal.biohpc.swmed.edu/intranet/terminal/ondemand_rstudio/

R packages installation

Where are your R packages installed?

- First time users will be asked to allow R to install packages into your home directory

```
> .libPaths()
1  "/home2/<username>/R/x86_64-pc-linux-gnu-library/3.6" #your local directory
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
```

Depending on packages, commands can be

- install.packages('devtools')
- install_github("danielwilhelm/cats")
- BiocManager::install("EnhancedVolcano")

Missing dependency

lib***.so.*: cannot open shared object file: No such file or directory

- Check the module list, often the dependency is installed as a module, using:
  module avail <package name>, e.g, geos, proj, hdf5, gdal
  then **module load <package name>** and redo installation

  - In rare cases, you will also need to add some options in the command

remotes::install_github("r-spatial/sf", configure.args="--with-proj-include=/cm/shared/apps/proj/gcc/6.0.0/include --with-proj-lib=/cm/shared/apps/proj/gcc/6.0.0/lib/ --with-proj-api=yes")

- This works for R version <= 4.1.1

Missing dependency

- We start to manage R v4.2 on BioHPC in a different way
- It was built inside a container, which makes it easier to install missing libraries without affecting the whole system
- However, this requires root privilege, so feel free to let us know if you have difficulty installing any R packages in Ondemand Rstudio – R v4.2.2
- We noticed R v4.3.1 was released a few month ago and we are working on it

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

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=/path/to/install
make –j   <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      v/virtual/env
cd /path/t      new/v      l/env
python -      ve      env
source v      v/bin/a      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

Solution to “Can not connect ...” “Connection failed”

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')
- Sys.setenv(http_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 packages to your accessible locations

Cluster-wide installation is possible. Please submit your request to BioHPC Helpdesk (biohpc-help@utsouthwestern.edu)

Questions / Comments / Remarks ?