UW Tower Data Center

Research Computing Club Presents

Hyak Training Session

October 27, 2020

Jesse Prelesnik (Training Coordinator)

Outline

Where I assume you're at right now:

"I finished the steps to getting access* to Hyak, but don't really know what to do now. How do I get up and running with the computing I want to do?"

I. Hyak overview

A. Hyak architecture
B. Logging on to Hyak

II. Navigating Hyak

A. Basic commands
B. Important locations
C. Transferring files

III. Slurm

A. Running a job
B. Commands
C. Modules
D. Interactivity

IV. Other resources

A. Topics outside of this tutorial
B. Places to get help

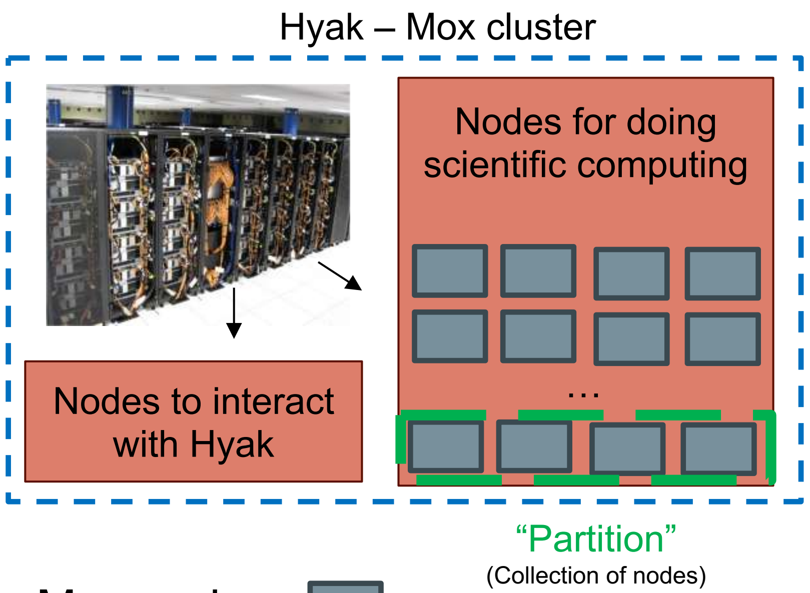
* https://depts.washington.edu/uwrcc/getting-started-2/getting-started/

Hyak overview

Hyak is a “condominium” supercomputing cluster:

- Research groups own nodes in partitions.
    - There’s one partition for all tuition-paying students:
    - Student Technology Fee – “`stf`”
- In addition to our partitions, groups have access to the `build` and `ckpt` partitions
- ~10,000 cores in total
    - A typical laptop might have 4-8 cores (CPUs)

One cluster for right now: “mox” (current), soon “clone” (future)
An old cluster, “ikt”, was just retired and is no longer usable


Hyak – Mox cluster
Nodes for doing scientific computing
…
Nodes to interact with Hyak
“Partition”
(Collection of nodes)

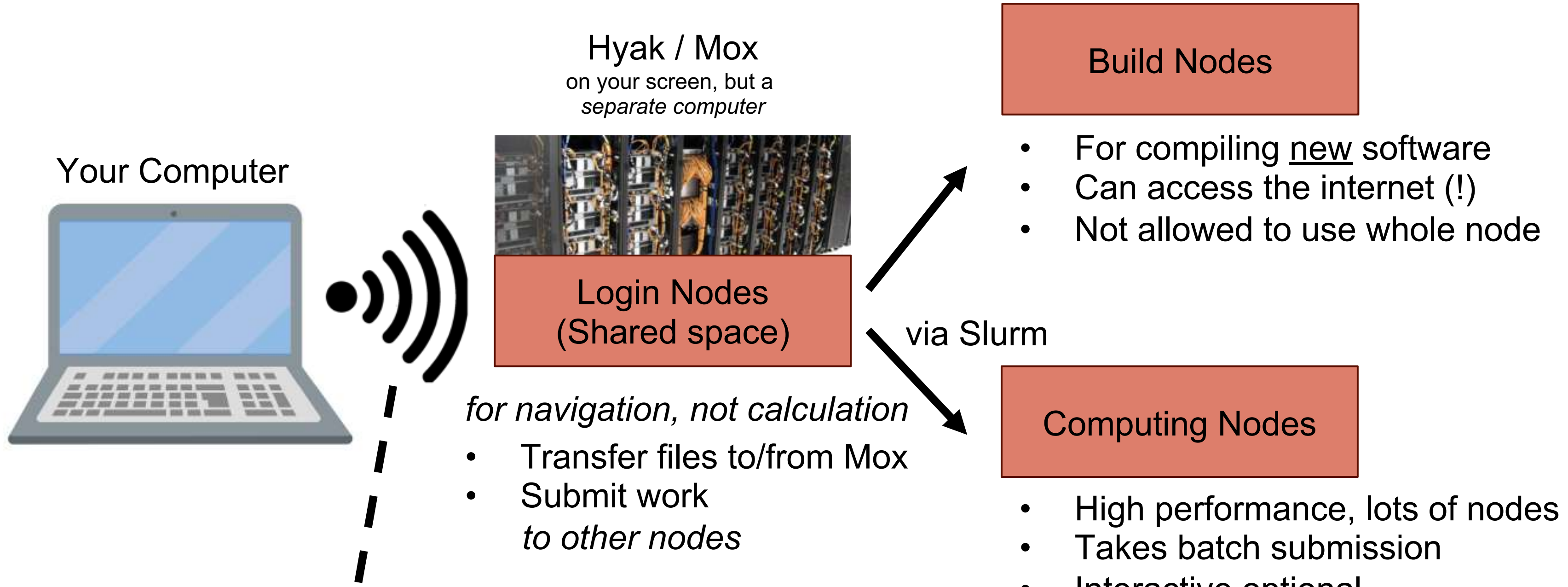
Mox nodes: ▨

- 28 cores
- 128 GB RAM
- 92 regular nodes - `stf` partition

parallel computing

Accessing Nodes - Hyak "Architecture"

- All nodes share the same filesystem (except `/tmp`)



for navigation, not calculation

- Transfer files to/from Mox
- Submit work

  to other nodes

Build Nodes:

- For compiling <u>new</u> software
- Can access the internet (!)
- Not allowed to use whole node

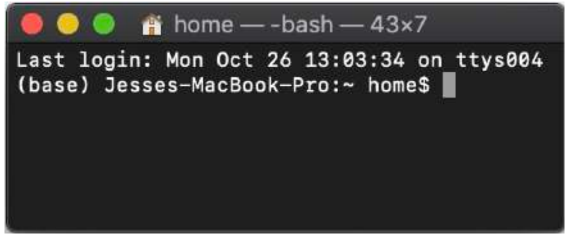
Computing Nodes:

- High performance, lots of nodes
- Takes batch submission
- Interactive optional
- Actual work is done here

login via `ssh` command
(next slide)

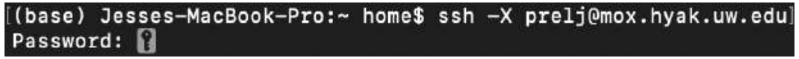
Logging on to Hyak

Terminal – “command line”
Easily accessible on Mac/Linux



```
home — -bash — 43×7
Last login: Mon Oct 26 13:03:34 on ttys004
(base) Jesses-MacBook-Pro:~ home$ 
```

“Secure Shell” ssh command
Enables connection to remote machine

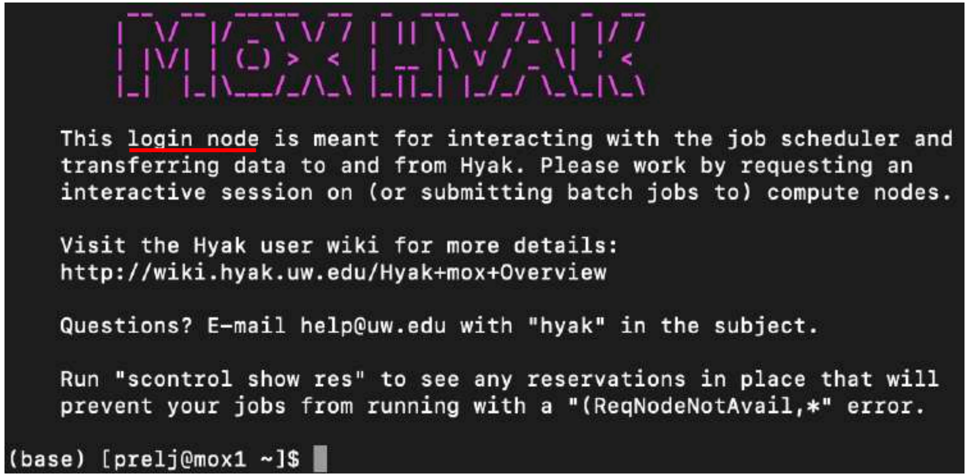


```
(base) Jesses-MacBook-Pro:~ home$ ssh -X prelj@mox.hyak.uw.edu
Password:
```

`ssh <uwnetid>@mox.hyak.uw.edu`

Two-factor authentication required
(Through UW-IT)

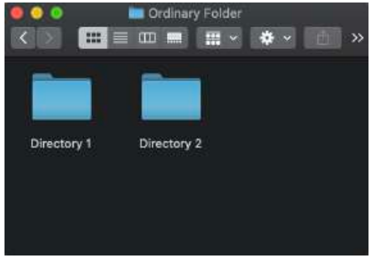
You’re in!



```
This login node is meant for interacting with the job scheduler and
transferring data to and from Hyak. Please work by requesting an
interactive session on (or submitting batch jobs to) compute nodes.

Visit the Hyak user wiki for more details:
http://wiki.hyak.uw.edu/Hyak+mox+Overview

Questions? E-mail help@uw.edu with "hyak" in the subject.

Run "scontrol show res" to see any reservations in place that will
prevent your jobs from running with a "(ReqNodeNotAvail,*" error.

(base) [prelj@mox1 ~]$ 
```

- Notice you are defaulted to the *login node*
- Jobs are not run here

“Graphical user interface” (GUI)



No GUI
Navigate via command line

What if I’m on Windows?

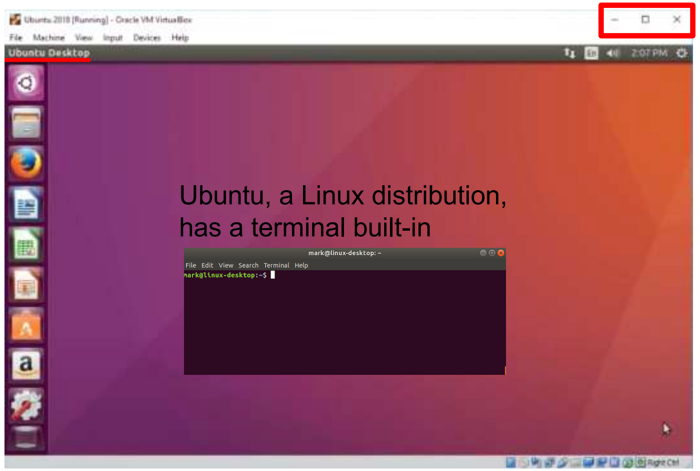
You’ll need a terminal somehow, and Windows does not provide a good one by default

Oracle VM VirtualBox

My preferred route: “Virtual Machine” (VM)
Sets up a mini environment that emulates a different operating system
Lets your Windows machine have some non-Windows capabilities (like terminal)

“Oracle” is one example of a VM

New window pops up within Windows to interface with Ubuntu


Ubuntu 2018 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Ubuntu Desktop
2:07 PM
mark@linux-desktop: ~
File Edit View Search Terminal Help
Right Ctrl

```
mark@linux-desktop:~$
```

Ubuntu, a Linux distribution, has a terminal built-in

Other options for Windows:

- PuTTY (Terminal emulator)
- CygWin (Runtime environment)
- GitBash
- Windows Subsytem for Linux
- WinSCP (just for transfering files)
- cmder
- xshell
- … and probably more!

How you get a terminal does not matter, they are all effectively the same

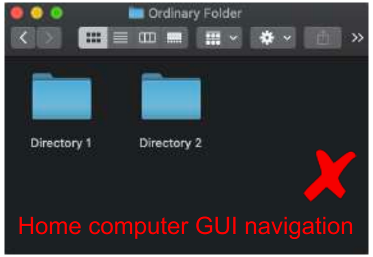
More help:
http://wiki.cac.washington.edu/display/hyakusers/Logging+In

I’m in

… Now what?

Note: On a computer, a "folder" and a "directory" are the same thing

Exploring Hyak



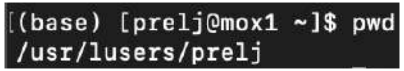
Hyak file systems through **command line**

```
(base) [prelj@mox1 ~]$ 
```

Essential First Actions:

- `mkdir`
  - "Make directory"
  - Create a place to put your files
- `vim, emacs, nano` (etc.)
  - Command line text editors

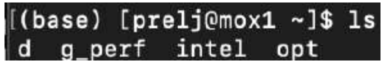
Where in the computer am I?
pwd – "Print working (current) directory"



```
(base) [prelj@mox1 ~]$ pwd
/usr/lusers/prelj
```

My home directory (see next slide)

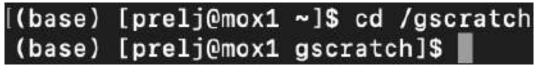
What folders and files are here?
`ls` – "List" files in current directory (folder)



```
(base) [prelj@mox1 ~]$ ls
d  g_perf  intel  opt
```

Each research group has a folder in `gscratch`

Let's go to where everybody stores files
cd – "Change directory"



```
(base) [prelj@mox1 ~]$ cd /gscratch
(base) [prelj@mox1 gscratch]$ 
```

Important locations on Hyak (get there with cd)

- ⭐ `/gscratch/stf`
  - Main work location for stf users
  - **Any files untouched for >30 days will be auto-deleted!!**
- ⭐ `/usr/lusers/<username>`
  - Home directory "~", you are put here by default
  - **Only 10 GB of storage per user**
- `/tmp`
  - Node local storage (separate from shared filesystem)
- `/sw`
  - All software installs
- `/sw/contrib`
  - User installed software
- `/sw/modules-1.775/modulefiles/contrib`
  - User added modulefiles

Where pre-installed programs (and their supporting files) are housed

You will seldom need to go here

Basic shell (bash) commands: Try them out!

File system manipulation:

- `ls`
  - “List” files in current directory (folder)
- `cd`
  - “Change directory”
- `pwd`
  - “Print working (current) directory”
- `mkdir`
  - “Make directory”
- `mv`
  - Move (rename) file or directory
- `cp`
  - “Copy” files and/or directories (-r)
- `rm`
  - “Remove” files and/or directories (-r)

File editing and compression:

- `nano`
  - Edit files
  - Other editors: vim, emacs, etc.
- `tar`
  - Compress for a “tape archive”
- `zip` (and `unzip`)
  - Compress via zip algorithm
  - Windows friendly

Many, many more

- **`man`**
- `find`
- `top`
- `kill`
- `chmod`
- `curl`
- `grep`
- `sed`

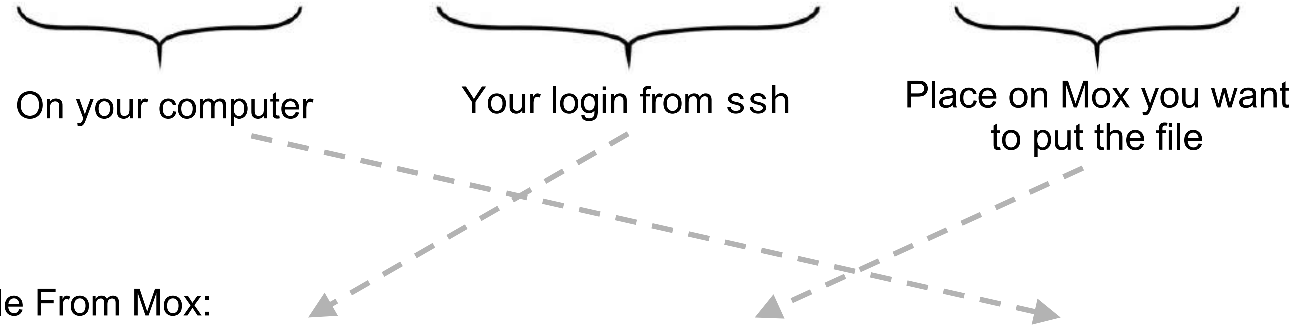
`...`

Transferring files

`scp` – Secure Copy

- Send file To Mox:

```
scp <path/to/file> <username>@mox.hyak.uw.edu:<path/to/dest>
```



- Get file From Mox:

```
scp <username>@mox.hyak.uw.edu:<path/to/file> <path/to/dest>
```

Flag `-r` needed to copy whole folder at once
An alternative copier for synchronizing directories is `scopy`

Lolo:

- Magnetic tape archive (lolo archive)
- For long term storage - **only store compressed large files!**
- STF location: `/archive/hyak/stf`
- Transfer files the same as between local and Hyak

Running Your First Job

Real computing jobs are passed to Slurm

- Ensures fair share between users
- Run interactive or batch jobs
- Allows for running on the `ckpt` queue
    - (more on this later)

Some information is needed during job setup:

<u>Partition</u>: whose CPUs/GPUs are we using? (we'll use stf)
<u>Allocation</u>: which bank account pays for these? (also stf)
<u>Locations</u>: where to write/read files to/from
<u>Resources</u>: how many CPUs/GPUs are needed, and for how long?
<u>Modules</u>: which programs should be loaded to use?

$\Big\}$ Packaged into a "Slurm script"

slurm
workload manager

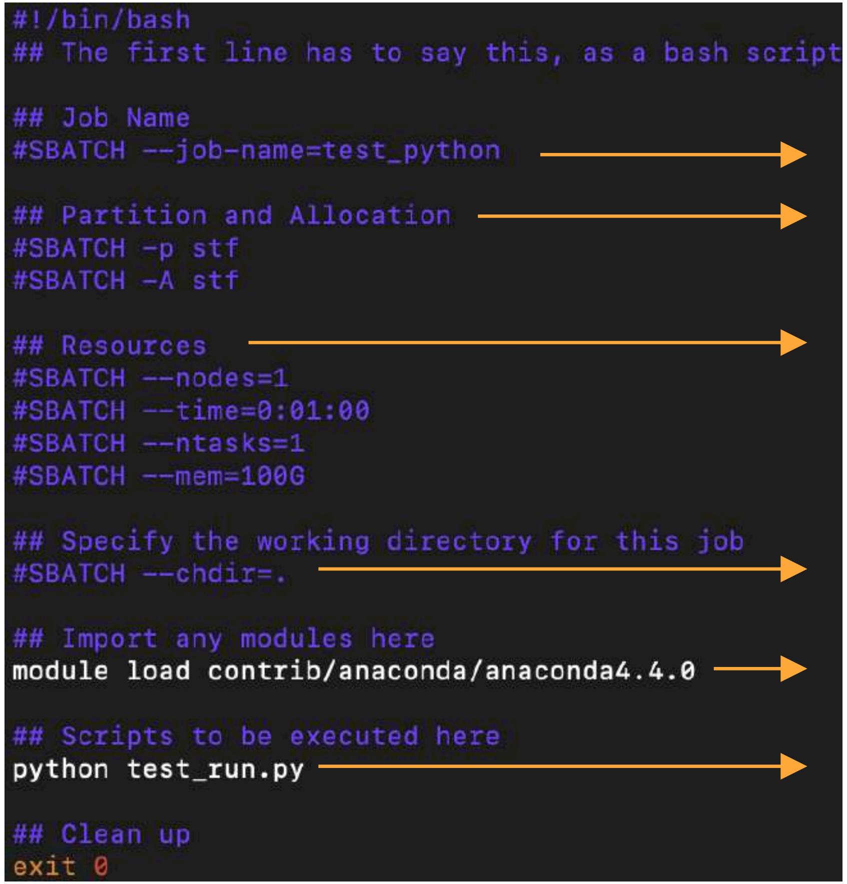
https://slurm.schedmd.com/

"Job Scheduler"

Manages multi-node jobs and communication protocols

Anatomy of a slurm script (written in “bash”)

In a text file called “submit.slurm”



```
#!/bin/bash
## The first line has to say this, as a bash script

## Job Name
#SBATCH --job-name=test_python

## Partition and Allocation
#SBATCH -p stf
#SBATCH -A stf

## Resources
#SBATCH --nodes=1
#SBATCH --time=0:01:00
#SBATCH --ntasks=1
#SBATCH --mem=100G

## Specify the working directory for this job
#SBATCH --chdir=.

## Import any modules here
module load contrib/anaconda/anaconda4.4.0

## Scripts to be executed here
python test_run.py

## Clean up
exit 0
```

Any line that begins with ## is a comment and is not read
Each line that begins with #SLURM specifies info for slurm
The #SLURM options can be provided in any order

- Name is optional
- Use stf nodes from the stf budget
- Request (1) node
- Time – Hours : Minutes : Seconds
- A single node has 28 cores and 128 GB memory
- This dummy job is not parallel, so only (1) core
- Directory “.” is *current location* to read/write files (the same place submit.slurm is saved)
- Load in a pre-installed python package
- The real run command

To execute this job,

```
sbatch submit.slurm
```

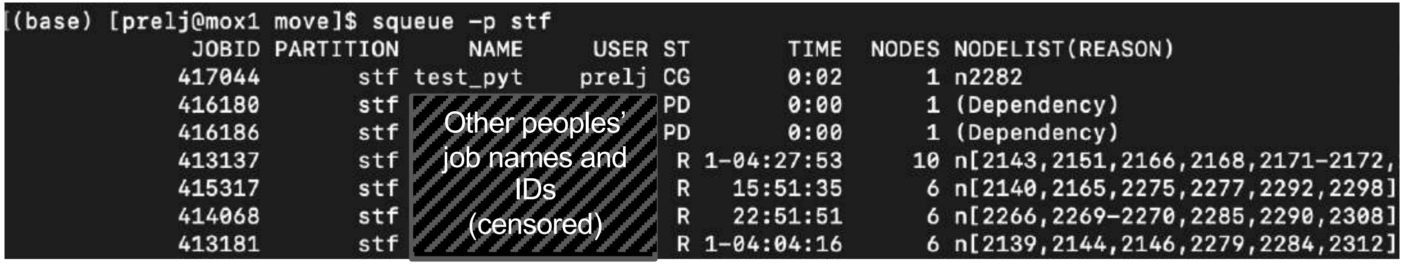
Then We Wait

- `sbatch <script>`
  - Submits a script for non-interactive use
  - Used this to submit a job on previous slide

Want to check the status?

- `squeue`
  - Flag –u `<uwnetid>` for only your jobs
  - Specify –p or –A for whole queue

What it looks like to check queue:



```
[(base) [prelj@mox1 move]$ squeue -p stf
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            417044       stf test_pyt    prelj CG       0:02      1 n2282
            416180       stf [redacted]          PD       0:00      1 (Dependency)
            416186       stf                   PD       0:00      1 (Dependency)
            413137       stf                    R 1-04:27:53     10 n[2143,2151,2166,2168,2171-2172,
            415317       stf                    R   15:51:35      6 n[2140,2165,2275,2277,2292,2298]
            414068       stf                    R   22:51:51      6 n[2266,2269-2270,2285,2290,2308]
            413181       stf                    R 1-04:04:16      6 n[2139,2144,2146,2279,2284,2312]
```

Other peoples' job names and IDs (censored)

Something wrong? / Taking too long?

- `scancel <jobid>`
  - Cancels an unfinished job
  - Can only cancel your own

Always check your output!

JOBID is used to refer to specific jobs
NODELIST has ID #s for individual nodes being used
ST column is status

R – Running (currently active)
PD – Pending (in queue)
CG – Completing (in process of termination)

Slurm: Commands

- `sbatch <script>`
  - Submits a script for non-interactive use
- `squeue`
  - Get status of jobs in batch queue
- `scancel <jobid>`
  - Cancels an unfinished job

I use these most often

- `srun`
  - Submits job for interactive use or initiate job steps inside batch script
  - (More on interactive jobs in a minute)
- `sinfo`
  - Get state of partitions and nodes (is the system operational)
- `sacct`
  - Gets accounting information about active and completed jobs

Slurm docs and `man <slurm-command>` are very useful!

("man" for manual)

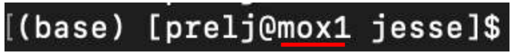
Running Jobs Interactively (command line, not script)

```
srun -p stf -A stf --ntasks=8 --mem=10G --time=0:10:00 --pty bash -l
```

- Partition
- Account
- Number of processes (*)
- Amount of RAM
- Time
- Command

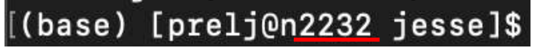
- Much of the same specifications as submit.slurm
- `--pty bash -l` signifies *interactive*
- Now you have command line control *while* accessing a compute node

Before running interactive job command



```
(base) [prelj@mox1 jesse]$
```

After running interactive job command



```
(base) [prelj@n2232 jesse]$
```

- Interactive nodes are computing nodes. You ***can*** submit jobs to other nodes from interactive nodes.

Be aware of difference between:

- Number of tasks (processes, MPI)
  - `--ntasks (-n)`
  - `--ntasks-per-node`
- Number of cpus per task (threads, OpenMP)
  - `--cpus-per-task` (-c)

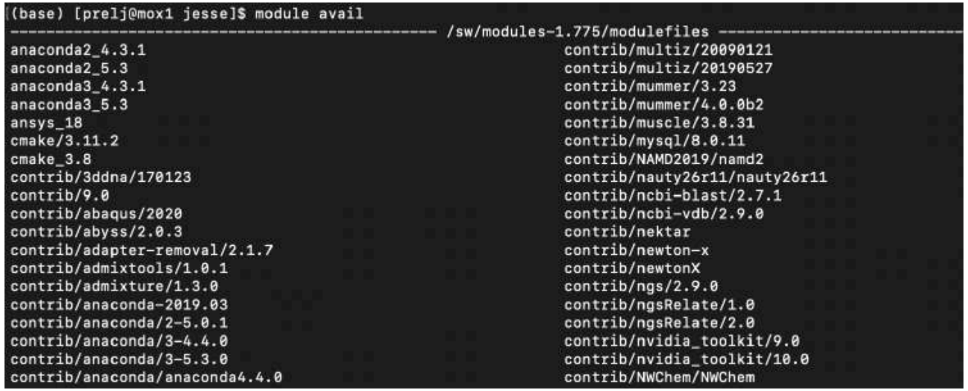
https://slurm.schedmd.com/srun.html

`man srun`

Loading software: the `modules` system

- `module avail`
    - Show all available modules
- `module load <module>`
- `module unload <module>`
    - (Un)load a given module
    - Provide full module name
- `module list`
    - List loaded modules
- `module purge`
    - Unload all modules
- `module help`
    - Print help with commands

Running `module avail`



```
(base) [prelj@mox1 jesse]$ module avail
--------------------------------------------- /sw/modules-1.775/modulefiles ---------------------------
anaconda2_4.3.1                                            contrib/multiz/20090121
anaconda2_5.3                                              contrib/multiz/20190527
anaconda3_4.3.1                                            contrib/mummer/3.23
anaconda3_5.3                                              contrib/mummer/4.0.0b2
ansys_18                                                   contrib/muscle/3.8.31
cmake/3.11.2                                               contrib/mysql/8.0.11
cmake_3.8                                                  contrib/NAMD2019/namd2
contrib/3ddna/170123                                       contrib/nauty26r11/nauty26r11
contrib/9.0                                                contrib/ncbi-blast/2.7.1
contrib/abaqus/2020                                        contrib/ncbi-vdb/2.9.0
contrib/abyss/2.0.3                                        contrib/nektar
contrib/adapter-removal/2.1.7                              contrib/newton-x
contrib/admixtools/1.0.1                                   contrib/newtonX
contrib/admixture/1.3.0                                    contrib/ngs/2.9.0
contrib/anaconda-2019.03                                   contrib/ngsRelate/1.0
contrib/anaconda/2-5.0.1                                   contrib/ngsRelate/2.0
contrib/anaconda/3-4.4.0                                   contrib/nvidia_toolkit/9.0
contrib/anaconda/3-5.3.0                                   contrib/nvidia_toolkit/10.0
contrib/anaconda/anaconda4.4.0                             contrib/NWChem/NWChem
```

… and more

> 400 modules available on Hyak right now!
Useful programs you'd use on your own computer, i.e.

- Python/Anaconda, Gromacs, NAMD, R, Mathematica, Matlab, Gaussian, compilers and more

Parallel computing tools

- CUDA, IMPI, etc.

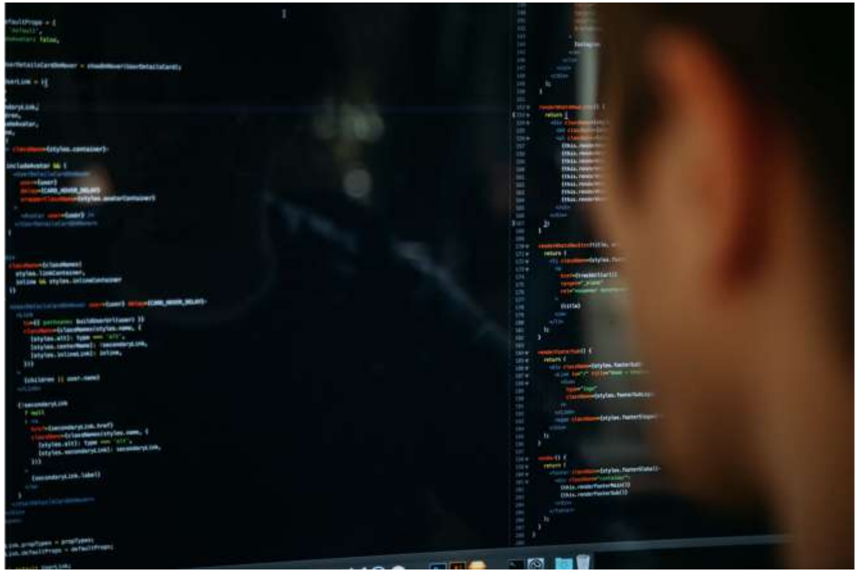
Advanced user's note:
The modules system works by keeping track of and modifying environment variables (e.g. PATH, LD_LIBRARY_PATH, CPATH, etc.)
https://modules.readthedocs.io/en/latest/

Checkpoint queue – special case

The checkpoint queue allows any user to run on other groups' unused nodes!

- Partition: `ckpt`
- Account: `<group>-ckpt` (`stf-ckpt`)
- Jobs can be interrupted at any time
- Jobs *will* be interrupted after 4 contiguous hours
- Jobs will be resubmitted after interruption (if under total requested time limit)
- **Your code must be checkpointed to take advantage of this**
  - Save a binary file containing state of some objects, restart from logs, etc.
  - Your script should also account for any checkpointing that is done

Topics not covered



- How to install new software
  - Installation and build systems
  - Writing your own modulefiles
- How to parallelize your code
  - Different paradigms for parallelism
  - Data locality and contiguity
  - GPU parallelization
  - Parallel patterns
  - Some programs have parallelization options built-in
- Anything with the cloud (This is the Hyak training session!)
- General architecture of HPC systems
  - Interconnectivity and node locality
  - Physical infrastructure

Where to get help

- Documentation (`man` or webpages)
- Hyak wiki: https://wiki.cac.washington.edu/display/hyakusers/WIKI+for+Hyak+users
- Slack channel: https://uw-rcc.slack.com/
- Website: https://depts.washington.edu/uwrcc/
- Emails: hpcc@uw.edu or uwrcc@uw.edu
- Office hours: Fridays from 1-3 pm

Happy computing!