Intro to HPC @ TACC Documentation

Release 0.0.1

Texas Advanced Computing Center

Jun 29, 2020

Contents:

1 Introduction to High Performance Computing 1

2 Getting Set Up 5

3 Introduction to the Command Line 9

4 Running Jobs 21

5 Indices and tables 29

CHAPTER 1

Introduction to High Performance Computing

In this session, we will learn the basics of high performance computing (HPC) at TACC, including system architecture, TACC HPC system resources, and best practices when using TACC systems.

1.1 Objectives for this Session

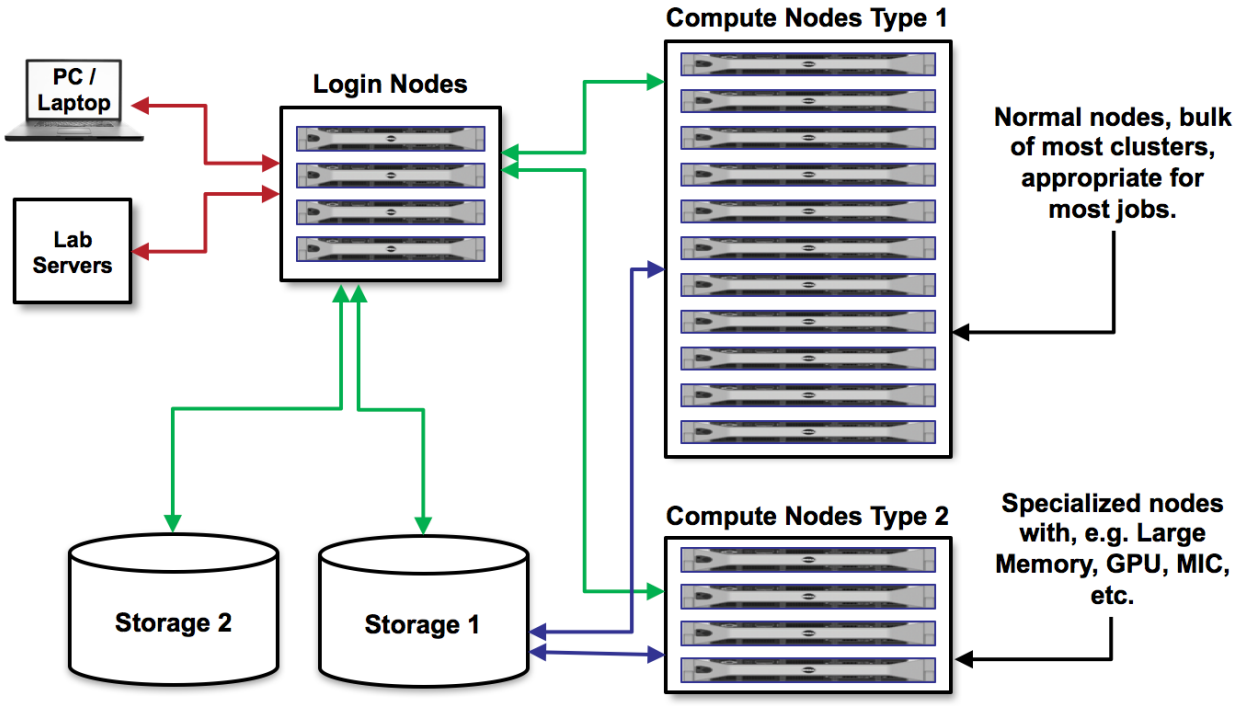
- Learn basic HPC system architecture
- Learn the difference between login and compute nodes
- Learn best practices when using a shared resource
- Learn about the disk/file system resources on TACC HPC systems

1.1.1 Introduction to High Performance Computing

Basic High Performance Computing (HPC) System Architecture

As you prepare to use TACC systems for this institute, it is important to understand the basic architecture. Think of an HPC resource as a very large and complicated lab instrument. Users need to learn how to:

- Interface with it / push the right buttons (Linux)
- Load samples (data)
- Run experiments (jobs)
- Interpret the results (data analysis / vis)



Login vs. Compute Nodes

As we've discussed, an HPC system has login nodes and compute nodes. We cannot run applications on the login nodes because they require too many resources and will interrupt the work of others. Instead, we must submit a job to a queue to run on compute nodes.

Tips for Success

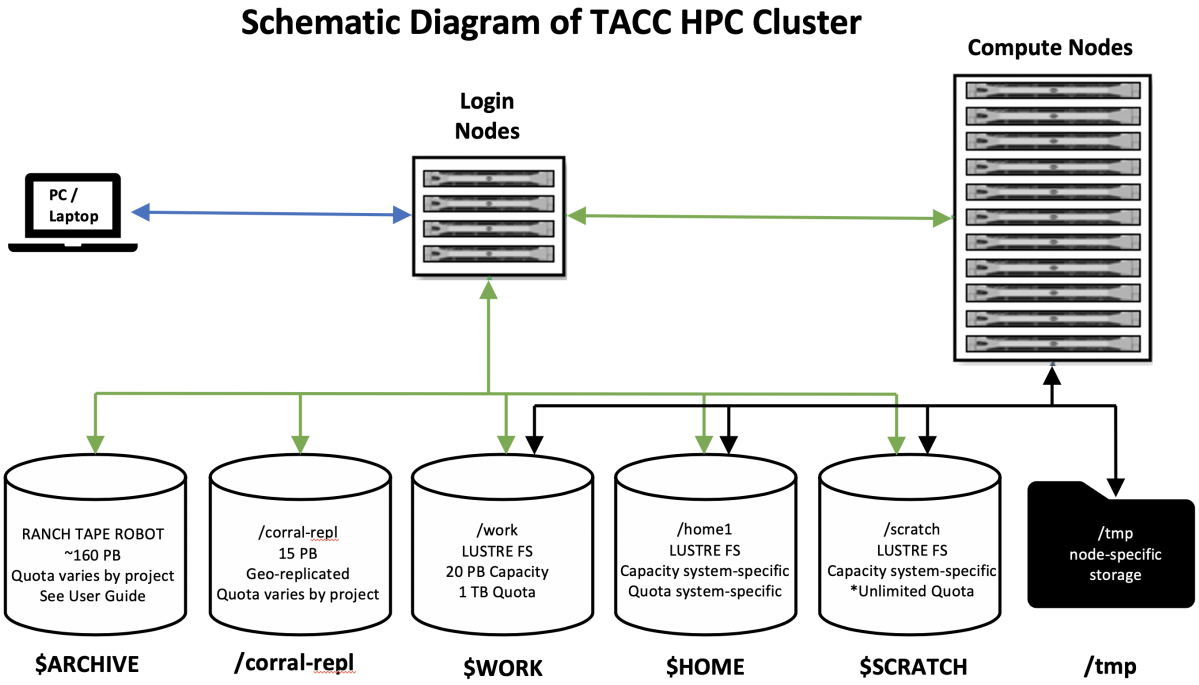
Read the documentation.

- Learn node schematics, limitations, file systems, rules
- Learn about the scheduler, queues, policies
- Determine the right resource for the job

User Responsibility on Shared Resources

HPC systems are shared resources. Your jobs and activity on a cluster, if mismanaged, can affect others. TACC staff are always available to help.

1.1.2 What are the disk/file system resources on TACC HPC systems?



A typical HPC cluster schematic at TACC is outlined above.

For example, Stampede2 uses:

/home1

- LustreFS
- ~1 PB overall capacity
- each user has 10 GB quota
- ENV VAR: `$HOME`

/scratch

- LustreFS
- ~30 PB overall capacity
- Unlimited quota, but 10 day limit
- ENV VAR: `$SCRATCH`

/work

- LustreFS
- 20 PB global share work file system
- each user has 1 TB quota
- ENV VAR: `$WORK`

/tmp

Each compute node has a local `/tmp` directory. `/tmp` can be used to read/write files that do not need to be accessed by other tasks. Data stored in `/tmp` is temporary, and lasts only for the duration of your job.

- Stampede2 SKX: 144 GB storage per compute node
- Stampede2 KNL: 107 GB storage per compute node for `normal`/`large`, 32 GB storage per compute node for `development`

TACC HPC Storage Systems

/corral-repl

- GPFS/NFS
- 6x2 PB
- 6 PB in each of Austin and Arlington
- typical quota: 5 TB but varies based on need
- folder path provided upon allocation (ex. `/corral-repl/TACC/bio/ECR`)

RANCH TAPE Robot

- ~160 PB backing tape store
- quota varies based on need, though users are typically limited to 2 TB
- ENV VAR: `$ARCHIVER`, `$ARCHIVE`

Stockyard

Global File System

- The `$WORK` filesystem on Stockyard helps knit TACC HPC Systems together
- Files on `$WORK` are present on most systems
- 20 PB of storage capacity
- 100+ gigabytes per second of aggregate bandwidth

CHAPTER 2

Getting Set Up

To log in to Stampede2, follow the instructions for your operating system below.

2.1 Mac / Linux

```
Open the application 'Terminal'
ssh username@stampede2.tacc.utexas.edu
(enter password)
(enter 6-digit token)
```

2.2 Windows

Windows users will need to install **PuTTY** to follow along. If you have not done so already, download the **PuTTY** "Windows Installer" here.

Once **PuTTY** is installed:

- Double-click the **PuTTY** icon
- In the **PuTTY** Configuration window make sure the Connection type is SSH
- enter `stampede2.tacc.utexas.edu` for Host Name
- click "Open"
- answer "Yes" to the SSH security question

In the **PuTTY** terminal:

- enter your TACC user id after the "login as:" prompt, then Enter
- enter the password associated with your TACC account
- enter your 6-digit TACC token value

```
Open the application 'PuTTY'
enter Host Name: stampede2.tacc.utexas.edu
(click 'Open')
(enter username)
(enter password)
(enter 6-digit token)
```

2.3 Successful Login to Stampede2

If your login was successful, your terminal will look something like this:

```
--------------------------------------------------------------------------------
                    Welcome to the Stampede2 Supercomputer
      Texas Advanced Computing Center, The University of Texas at Austin
--------------------------------------------------------------------------------

           ** Unauthorized use/access is prohibited. **

If you log on to this computer system, you acknowledge your awareness
of and concurrence with the UT Austin Acceptable Use Policy. The
University will prosecute violators to the full extent of the law.

TACC Usage Policies:
http://www.tacc.utexas.edu/user-services/usage-policies/
________________________________________________________________________________

Welcome to Stampede2, *please* read these important system notes:

--> Stampede2 user documentation is available at:
       https://portal.tacc.utexas.edu/user-guides/stampede2

---------------------- Project balances for user ancantu -----------------------
| Name           Avail SUs     Expires | Name           Avail SUs     Expires |
| Project1           #####  20XX-0X-XX | Project3            ####  20XX-0X-XX |
| Project2           #####  20XX-0X-XX | Project4            ####  20XX-0X-XX |
------------------------ Disk quotas for user ancantu ------------------------
| Disk         Usage (GB)     Limit    %Used   File Usage       Limit    %Used |
| /home1             0.0      10.0     0.00           23      200000     0.01 |
| /work             12.6    1024.0     1.23         3131     3000000     0.10 |
| /scratch           0.0       0.0     0.00            4           0     0.00 |
--------------------------------------------------------------------------------
```

2.4 A Note About Quotas

The welcome message you receive upon successful login to Stampede2 has useful information for you to keep track of. Especially of note is the breakdown of disk quotas for your account, as you can keep an eye on whether your usage is nearing the determined limit.

Once your usage is nearing the quota, you'll start to experience issues that will not only impact your own work, but also impact the system for others. For example, if you're nearing your quota in `$WORK`, and your job is repeatedly trying (and failing) to write to `$WORK`, you will stress that file system.

Another useful way to monitor your disk quotas (and TACC project balances) at any time is to execute:

```
login1$ /usr/local/etc/taccinfo # Generally more current than balances displayed on
↪the portals.
```

CHAPTER 3

Introduction to the Command Line

3.1 Set Up Reminder

To log in to Stampede2, follow the instructions for your operating system below.

Mac / Linux

```
Open the application 'Terminal'
ssh username@stampede2.tacc.utexas.edu
(enter password)
(enter 6-digit token)
```

Windows

```
Open the application 'PuTTY'
enter Host Name: stampede2.tacc.utexas.edu
(click 'Open')
(enter username)
(enter password)
(enter 6-digit token)
```

3.2 Module Objectives

This module will be fully interactive. Participants are **strongly encouraged** to follow along on the command line. Even if you already have command line familiarity, please follow along because we will create files / directories that we use later on in the session. After completing this module, participants should be able to:

- Describe basic functions of essential Linux commands
- Use Linux commands to navigate a file system and manipulate files
- Edit files directly on a Linux system using a command line utility (e.g. vim, nano, emacs)
- Transfer data to a remote Linux file system

- Print, identify, and modify environment variables

3.3 Topics Covered

- Creating and changing folders (`pwd`, `ls`, `mkdir`, `cd`, `rmdir`)
- Creating and manipulating files (`touch`, `rm`, `mv`, `cp`)
- Looking at the Contents of files (`cat`, `more`, `less`, `head`, `tail`, `grep`)
- Text editing with vim (insert mode, normal mode, navigating, saving, quitting)
- Network and file transfers (`hostname`, `whoami`, `logout`, `ssh`, `scp`, `rsync`)

3.3.1 Creating and Changing Folders

On a Windows or Mac desktop, our present location determines what files and folders we can access. I can "see" my present location visually with the help of the graphic interface - I could be looking at my Desktop, or the contents of a folder, for example. In a Linux command-line interface, we lack the same visual queues to tell us what our location is. Instead, we use a command - `pwd` (print working directory) - to tell us our present location. Try executing this command on Stampede2:

```
$ pwd
/home1/03439/wallen
```

This home location on the Linux filesystem is unique for each user, and it is roughly analogous to C:\Users\username on Windows, or /Users/username on Mac.

To see what files and folders are available at this location, use the `ls` (list) command:

```
$ ls
```

I have no files or folders in my home directory yet, so I do not get a response. We can create some folders using the `mkdir` (make directory) command. The words 'folder' and 'directory' are interchangeable:

```
$ mkdir folder1
$ mkdir folder2
$ mkdir folder3
```

```
$ ls
folder1 folder2 folder3
```

Now we have some folders to work with. To "open" a folder, navigate into that folder using the `cd` (change directory) command. This process is analogous to double-clicking a folder on Windows or Mac:

```
$ pwd
/home/03439/wallen/
$ cd folder1
$ pwd
/home1/03439/wallen/folder1
```

Now that we are inside `folder1`, make a few sub-folders:

```
$ mkdir subfolderA
$ mkdir subfolderB
$ mkdir subfolderC
$ ls
subfolderA subfolderB subfolderC
```

Use `cd` to Navigate into `subfolderA`, then use `ls` to list the contents. What do you expect to see?

```
$ cd subfolderA
$ pwd
/home/03439/wallen/folder1/subfolderA
$ ls
```

There is nothing there because we have not made anything yet. Next, we will navigate back to the home directory. So far we have seen how to navigate "down" into folders, but how do we navigate back "up" to the parent folder? There are different ways to do it. For example, we could specify the complete path of where we want to go:

```
$ pwd
/home1/03439/wallen/folder1/subfolderA
$ cd /home1/03439/wallen/folder1
$ pwd
/home1/03439/wallen/folder1/
```

Or, we could use a shortcut, `..`, which refers to the **parent folder** - one level higher than the present location:

```
$ pwd
/home1/03439/wallen/folder1
$ cd ..
$ pwd
/home1/03439/wallen
```

We are back in our home directory. Finally, use the `rmdir` (remove directory) command to remove folders. This will not work on folders that have any contents (more on this later):

```
$ mkdir junkfolder
$ ls
folder1 folder2 folder3 junkfolder
$ rmdir junkfolder
$ ls
folder1 folder2 folder3
```

A bonus command available on some Linux operating systems is called `tree`. The `tree` command displays files and folders in a hierarchical view. Use another Linux shortcut, `.`, to indicate that you want to list files and folders in your **present location**.

```
$ tree .
.
|-- folder1
|   |-- subfolderA
|   |-- subfolderB
|   `-- subfolderC
|-- folder2
`-- folder3
```

Before we move on, let's remove the directories we have made, using `rm -r` to remove our parent folder `folder1` and its subfolders. The `-r` command line option recursively removes subfolders and files located "down" the parent directory. `-r` is required for non-empty folders.

```
$ rm -r folder1
$ ls
folder2 folder3
```

Which command should we use to remove `folder2` and `folder3`?

```
$ rmdir folder2
$ rmdir folder3
$ ls
```

Why could we use `rmdir` on `folder2` and `folder3`, but not on `folder1`?

Exercise

1. Navigate to your home directory
2. Make a new folder called `challenge01`
3. Navigate into that new folder
4. Make 5 sub folders called `a`, `b`, `c`, `d`, `e`
5. Within each of those sub folders, make 5 sub folders called `1`, `2`, `3`, `4`, `5`
6. Navigate back to your home directory and print a hierarchical view of the `challenge01` folder

Review of Topics Covered

Command	Effect
`pwd`	print working directory
`ls`	list files and directories
`ls -l`	list files in column format
`mkdir dir_name/`	make a new directory
`cd dir_name/`	navigate into a directory
`rmdir dir_name/`	remove an empty directory
`rm -r dir_name/`	remove a directory and its contents
`tree`	list files and directories hierarchically
`.` or `./`	refers to the present location
`..` or `../`	refers to the parent directory

3.3.2 Creating and Manipulating Files

We have seen how to navigate around the filesystem and perform operations with folders. But, what about files? Just like on Windows or Mac, we can easily create new files, copy files, rename files, and move files to different locations. First, we will navigate to the home directory and create a few new folders and files with the `mkdir` and `touch` commands:

```
$ cd     # cd on an empty line will automatically take you back to the home directory
$ pwd
/home1/03439/wallen
$ mkdir folder1
$ mkdir folder2
$ mkdir folder3
```

(continues on next page)

(continued from previous page)

```
$ touch file_a
$ touch file_b
$ touch file_c
$ ls
file_a  file_b  file_c  folder1  folder2  folder3
```

These files we have created are all empty. Removing a file is done with the `rm` (remove) command. Please note that on Linux file systems, there is no "Recycle Bin". Any file or folder removed is gone forever and often un-recoverable:

```
$ touch junkfile
$ rm junkfile
```

Moving files with the `mv` command and copying files with the `cp` command works similarly to how you would expect on a Windows or Mac machine. The context around the move or copy operation determines what the result will be. For example, we could move and/or copy files into folders:

```
$ mv file_a folder1/
$ mv file_b folder2/
$ cp file_c folder3/
```

Before listing the results with `ls` or `tree`, try to guess what the result will be.

```
$ tree .
.
|-- file_c
|-- folder1
|   `-- file_a
|-- folder2
|   `-- file_b
`-- folder3
    `-- file_c
```

Two files have been moved into folders, and `file_c` has been copied - so there is still a copy of `file_c` in the home directory. Move and copy commands can also be used to change the name of a file:

```
$ cp file_c file_c_copy
$ mv file_c file_c_new_name
```

By now, you may have found that Linux is very unforgiving with typos. Generous use of the `<Tab>` key to auto-complete file and folder names, as well as the `<UpArrow>` to cycle back through command history, will greatly improve the experience. As a general rule, try not to use spaces or strange characters in files or folder names. Stick to:

```
A-Z      # capital letters
a-z      # lowercase letters
0-9      # digits
-        # hyphen
_        # underscore
.        # period
```

Before we move on, let's clean up once again by removing the files and folders we have created. Do you remember the command for removing non-empty folders?

```
$ rm -r folder1
$ rm -r folder2
$ rm -r folder3
```

How do we remove `file_c_copy` and `file_c_new_name`?

```
$ rm file_c_copy
$ rm file_c_new_name
```

Exercise

1. Navigate to your home directory
2. Execute this exact command: `cp -r /work/03439/wallen/public/challenge02 ./`
3. Navigate into the `challenge02` folder
4. Somewhere within there is a file. Can you find it?

Review of Topics Covered

Command	Effect
`touch file_name`	create a new file
`rm file_name`	remove a file
`rm -r dir_name/`	remove a directory and its contents
`mv file_name dir_name/`	move a file into a directory
`mv old_file new_file`	change the name of a file
`mv old_dir/ new_dir/`	change the name of a directory
`cp old_file new_file`	copy a file
`cp -r old_dir/ new_dir/`	copy a directory
`<Tab>`	autocomplete file or folder names
`<UpArrow>`	cycle through command history

3.3.3 Looking at the Contents of Files

Everything we have seen so far has been with empty files and folders. We will now start looking at some real data. Navigate to your home directory, then issue the following `cp` command to copy a "tar archive" file from me:

```
$ cd ~    # the tilde ~ is also a shortcut referring to your home directory
$ pwd
/home1/03439/wallen
$ cp /work/03439/wallen/public/IntroToLinuxHPC.tar .
```

Try to use `<Tab>` to autocomplete the name of the file. Do NOT change the username or lustre number to your username and lustre number - in this case you are copying a file from my directory to your directory. Also, please notice the single dot `.` at the end of the copy command, which indicates that you want to cp the tar archive to `.`, this present location (your home directory).

This archive file is actually a bundle of many files and folders, all packed into one nice, easily transportable object. Once it is copied over, un-pack the file with the following command:

```
$ ls
IntroToLinuxHPC.tar
$ tar -xvf IntroToLinuxHPC.tar
```

In the first lab (`Lab01`), we have a file called `websters.txt`. This a list of all the words in Webster's Dictionary. But how can we see the contents of the file?

```
$ cd IntroToLinuxHPC
$ cd Lab01
$ pwd
/home1/03439/wallen/IntroToLinuxHPC/Lab01
$ ls
README websters.txt
```

If you want to see the contents of a file, use the `cat` command to print it to screen:

```
$ cat websters.txt
A
a
aa
aal
aalii
aam
Aani
aardvark
```

This is a long file! Printing everything to screen is much too fast and not very useful. We can use a few other commands to look at the contents of the file with `more` control:

```
$ more websters.txt
```

Press the `<Enter>` key to scroll through line-by-line, or the `<Space>` key to scroll through page-by-page. Press `q` to quit the view, or `<Ctrl+c>` to force a quit if things freeze up. A `%` indicator at the bottom of the screen shows your progress through the file. This is still a little bit messy and fills up the screen. The `less` command has the same effect, but is a little bit cleaner:

```
$ less websters.txt
```

Scrolling through the data is the same, but now we can also search the data. Press the `/` forward slash key, and type a word that you would like to search for. The screen will jump down to the first match of that word. The `n` key will cycle through other matches, if they exist.

Finally, you can view just the beginning or the end of a file with the `head` and `tail` commands. For example:

```
$ head websters.txt
$ tail websters.txt
```

The > and >> shortcuts in Linux indicate that you would like to redirect the output of one of the commands above. Instead of printing to screen, the output can be redirected into a file:

```
$ cat websters.txt > websters_new.txt
$ head websters.txt > first_10_lines.txt
```

A single greater than sign > will redirect and **overwrite** any contents in the target file. A double greater than sign >> will redirect and **append** any output to the end of the target file.

One final useful way to look at the contents of files is with the `grep` command. `grep` searches a file for a specific pattern, and returns all lines that match the pattern. For example:

```
$ grep "banana" websters.txt
banana
cassabanana
```

Although it is not always necessary, it is safe to put the search term in quotes. More on `grep` later.

Exercise

1. Extract every word from `websters.txt` that contains the string `apple`, and put it into a new file called `apple.txt`.
2. Extract every word from `websters.txt` that contains the string `carrot`, and put it into a new file called `carrot.txt`.
3. Extract every word from `websters.txt` that contains the string `cheese`, and put it into a new file called `cheese.txt`.
4. Examine the contents of `apple.txt`, `carrot.txt`, and `cheese.txt` to make sure they contain what you expect.
5. Concatenate all three lists into a new file called `food.txt`.

Review of Topics Covered

Command	Effect
`cat file_name`	print file contents to screen
`cat file_name >> new_file`	redirect output to new file
`more file_name`	scroll through file contents
`less file_name`	scroll through file contents
`head file_name`	output beginning of file
`tail file_name`	output end of a file
`grep pattern file_name`	search for 'pattern' in a file
`~/`	shortcut for home directory
`<Ctrl+c>`	force interrupt
`>`	redirect and overwrite
`>>`	redirect and append

3.3.4 Text Editing with VIM

VIM is a text editor used on Linux file systems.

Open a file (or create a new file if it does not exist):

```
$ vim file_name
```

There are two "modes" in VIM that we will talk about today. They are called "insert mode" and "normal mode". In insert mode, the user is typing text into a file as seen through the terminal (think about typing text into TextEdit or Notepad). In normal mode, the user can perform other functions like save, quit, cut and paste, find and replace, etc. (think about clicking the menu options in TextEdit or Notepad). The two main keys to remember to toggle between the modes are `i` and `Esc`.

Entering VIM insert mode:

```
> i
```

Entering VIM normal mode:

```
> Esc
```

A summary of the most important keys to know for normal mode are (more on your cheat sheet):

```
# Navigating the file:

arrow keys        move up, down, left, right
    Ctrl+u        page up
    Ctrl+d        page down

         0        move to beginning of line
         $        move to end of line

        gg        move to beginning of file
         G        move to end of file
        :N        move to line N

# Saving and quitting:

        :q        quit editing the file
        :q!       quit editing the file without saving

        :w        save the file, continue editing
        :wq       save and quit
```

For more information, see:

- http://openvim.com/
- http://vim-adventures.com/
- Or type on the command line: `vimtutor`

Exercise

1. Navigate to the `Lab03` directory.
2. Open up the file called `tutorial.txt` and follow the instructions within.
3. Create a new file called `my_name.txt`, write your name within the file, save and quit, then print the contents of the file to screen.

Review of Topics Covered

Command	Effect
`vim file.txt`	open "file.txt" and edit with `vim`
`i`	toggle to insert mode
`<Esc>`	toggle to normal mode
`<arrow keys>`	navigate the file
`:q`	quit ending the file
`:q!`	quit editing the file without saving
`:w`	save the file, continue editing
`:wq`	save and quit

3.3.5 Network and File Transfers

In order to login or transfer files to a remote Linux file system, you must know the hostname (unique network identifier) and the username. If you are already on a Linux file system, those are easy to determine using the following commands:

```
$ whoami
wallen
$ hostname -f
login4.stampede2.tacc.utexas.edu
```

Given that information, a user would remotely login to this Linux machine using the Terminal command:

```
$ ssh wallen@stampede2.tacc.utexas.edu
(enter password)
(enter token)
```

Windows users would typically use the program **PuTTY** to perform this operation. Logging out of a remote system is done using the `logout` command, or the shortcut `<Ctrl+d>`:

```
[stampede2]$ logout
[local]$
```

To practice transferring files to Stampede2's `$WORK` and `$SCRATCH`, we need to identify the path to our `$WORK` and `$SCRATCH` directory. To identify these paths, we can use helpful command shortcuts.

To identify the path to our `$WORK` directory, we can use `cd $WORK` or the helpful shortcut `cdw`:

```
$ cdw
$ pwd
/work/03439/wallen/stampede2
```

To identify the path to our `$SCRATCH` directory, we can use `cd $SCRATCH` or the helpful shortcut `cds`:

```
$ cds
$ pwd
/scratch/03439/wallen
```

Copying files from your local computer to Stampede2's `$WORK` would require the `scp` command (Windows users use the program "WinSCP"):

```
[local]$ scp my_file wallen@stampede2.tacc.utexas.edu:/work/03439/wallen/stampede2
(enter password)
(enter token)
```

In this command, you specify the name of the file you want to transfer (`my_file`), the username (`wallen`), the hostname (`stampede2.tacc.utexas.edu`), and the path you want to put the file (`/work/03439/wallen/stampede2`). Take careful notice of the separators including spaces, the @ symbol, and the colon.

Copying files from your local computer to Stampede2's `$SCRATCH` using `scp`:

```
[local]$ scp my_file wallen@stampede2.tacc.utexas.edu:/scratch/03439/wallen
(enter password)
(enter token)
```

Copy files from Stampede2 to your local computer using the following:

```
[local]$ scp wallen@stampede2.tacc.utexas.edu:/work/03439/wallen/stampede2/my_file ./
(enter password)
(enter token)
```

Note: If you wanted to copy `my_file` from `$SCRATCH`, the path you would specify after the colon would be `/scratch/03439/wallen/my_file`.

Instead of files, full directories can be copied using the "recursive" flag (`scp -r ...`). The `rsync` tool is an advanced copy tool that is useful for synching data between two sites. Although we will not go into depth here, example `rsync` usage is as follows:

```
$ rsync -azv local remote
$ rsync -azv remote local
```

This is just the basics of copying files. See example `scp` usage here and example `rsync` usage here.

Exercise

1. Identify which Stampede2 login node you are on (login1, login2, login3)
2. Remotely login to a different Stampede2 login node and list what files are available.
3. Logout until you are back to your original login node.
4. Make your own `my_file` on your local computer using knowledge from our previous sections and copy `my_file` to your `$WORK` file system on Stampede2

Review of Topics Covered

Command	Effect
`hostname -f`	print hostname
`whoami`	print username
`ssh username@hostname`	remote login
`logout`	logout
`cd $WORK`, `cdw`	navigate to `$WORK` file system
`cd $SCRATCH`, `cds`	navigate to `$SCRATCH` file system
`scp local remote`	copy a file from local to remote
`scp remote local`	copy a file from remote to local
`rsync -azv local remote`	sync files between local and remote
`rsync -azv remote local`	sync files between remote and local
`<Ctrl+d>`	logout of host

CHAPTER 4

Running Jobs

4.1 Module Objectives

This module will be fully interactive. Participants are **strongly encouraged** to follow along on the command line. After taking this module, participants should be able to:

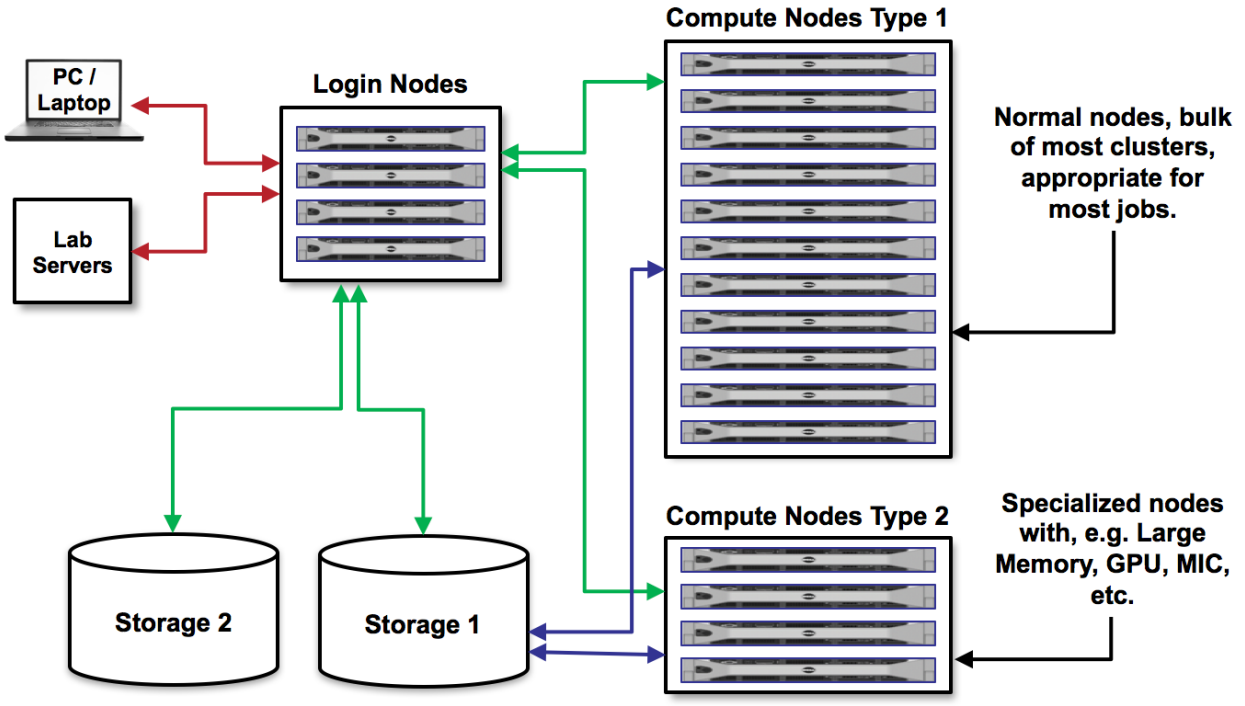
- Prepare and submit a batch job to a queue
- Initiate an interactive session using idev

4.2 Topics Covered

- Batch job submission, commands dependent on system (`sbatch`, `showq`, `scancel`).
- Interactive sessions with `idev` (`idev`, `idev -help`)

4.2.1 Batch Job Submission

As we discussed before, on Stampede2 there are login nodes and compute nodes.



We cannot run the applications we need for our research on the login nodes because they require too many resources and will interrupt the work of others. Instead, we must write a short text file containing a list of the resources we need, and containing the command(s) for running the application. Then, we submit that text file to a queue to run on compute nodes. This process is called **batch job submission**.

There are several queues available on Stampede2. It is important to understand the queue limitations and pick a queue that is appropriate for your job. Documentation can be found here. Today, we will be using the `development` queue which has a max runtime of 2 hours, and users can only submit one job at a time.

First, navigate to the `Lab04` directory where we have an example job script prepared, called `job.slurm`:

```
$ cd
$ cd IntroToLinuxHPC/Lab04
$ cat job.slurm

#!/bin/bash
#----------------------------------------------------
# Example SLURM job script to run applications on
# TACCs Stampede2 system.
#----------------------------------------------------
#SBATCH -J               # Job name
#SBATCH -o               # Name of stdout output file (%j expands to jobId)
#SBATCH -p               # Queue name
#SBATCH -N               # Total number of nodes requested (16 cores/node)
#SBATCH -n               # Total number of mpi tasks requested
#SBATCH -t               # Run time (hh:mm:ss)
#SBATCH -A               # <-- Allocation name to charge job against

# Everything below here should be Linux commands
```

First, we must know an application we want to run, and a research question we want to ask. This generally comes from your own research. For this example, we want to use the application called `autodock_vina` to check how well a small molecule ligand fits within a protein binding site. All the data required for this job is in a subdirectory

called `data`:

```
$ pwd
/home1/03439/wallen/IntroToLinuxHPC/Lab04
$ ls
data  job.slurm  results
$ ls data/
configuration_file.txt  ligand.pdbqt  protein.pdbqt
$ ls results/
     # nothing here yet
```

Next, we need to fill out `job.slurm` to request the necessary resources. I have some experience with `autodock_vina`, so I can reasonably predict how much we will need. When running your first jobs with your applications, it will take some trial and error, and reading online documentation, to get a feel for how many resources you should use. Open `job.slurm` with VIM and fill out the following information:

```
#SBATCH -J vina_job      # Job name
#SBATCH -o vina_job.o%j  # Name of stdout output file (%j expands to jobId)
#SBATCH -p development   # Queue name
#SBATCH -N 1             # Total number of nodes requested (16 cores/node)
#SBATCH -n 1             # Total number of mpi tasks requested
#SBATCH -t 00:10:00      # Run time (hh:mm:ss)
#SBATCH -A TRAINING-HPC  # <-- Allocation name to charge job against
```

Now, we need to provide instructions to the compute node on how to run `autodock_vina`. This information would come from the `autodock_vina` instruction manual. Continue editing `job.slurm` with VIM, and add this to the bottom:

```
# Everything below here should be Linux commands

echo "starting at:"
date

module list
module load intel/17.0.4
module load boost/1.64
module load autodock_vina/1.1.2
module list

cd data/
vina --config configuration_file.txt --out ../results/output_ligands.pdbqt

echo "ending at:"
date
```

The way this job is configured, it will print a starting date and time, load the appropriate modules, run `autodock_vina`, write output to the `results/` directory, then print the ending date and time. Keep an eye on the `results/` directory for output. Once you have filled in the job description, save and quit the file. Submit the job to the queue using the `sbatch` command‘:

```
$ sbatch job.slurm
```

To view the jobs you have currently in the queue, use the `showq` or `squeue` commands:

```
$ showq -u
$ showq        # shows all jobs by all users
$ squeue -u $USERNAME
$ squeue       # shows all jobs by all users
```

If for any reason you need to cancel a job, use the `scancel` command with the 6- or 7-digit jobid:

```
$ scancel jobid
```

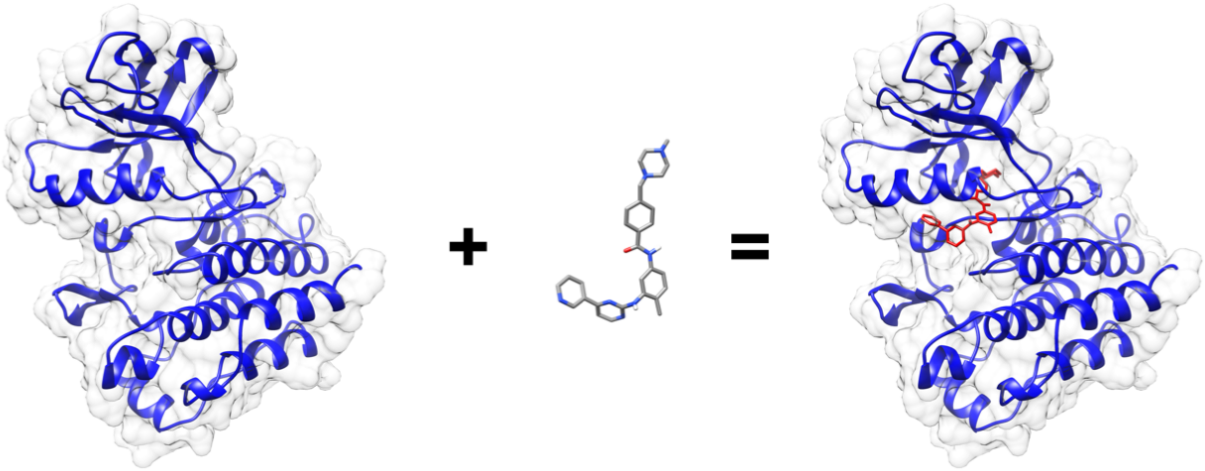
For more example scripts, see this directory on Stampede2:

```
$ ls /share/doc/slurm/
```

If everything went well, you should have an output file named something similar to `vina_job.o864828` in the same directory as the `job.slurm` script. And, in the `results/` directory, you should have some output:

```
$ cat vina_job.o864828
    # closely examine output

$ ls results
output_ligands.pdbqt
```



(Output visualized in UCSF Chimera)

Congratulations! You ran a batch job on Stampede2!

4.2.2 Interactive Sessions with idev

The `idev` utility initiates an interactive session on one or more compute nodes so that you can issue commands and run code as if you were doing so on your personal machine. An interactive session is useful for development, as you can quickly compile, run, and validate code. Accessing a single compute node is accomplished by simply executing `idev` on any of the TACC systems.

Initiating an Interactive Session

To learn about the command line options available for `idev`, use `idev -help`.

```
login1$ idev -help
...
OPTION ARGUMENTS          DESCRIPTION
-A     account_name       sets account name (default: in .idevrc)
-m     minutes            sets time in minutes (default: 30)
-p     queue_name         sets queue to named queue (default: -p development)
```

(continues on next page)

(continued from previous page)

```
-r     resource_name     sets hardware
-t     hh:mm:ss          sets time to hh:mm:ss (default: 00:30:00)
-help       [--help     ] displays (this) help message
-v          [--version  ] output version information and exit
```

Let's go over some of the most useful `idev` command line options that can customize your interactive session:

To change the **time** limit to be lesser or greater than the default 30 minutes, users can use the `-m` command line option. For example, a user requesting an interactive session for an hour would use the command line option `-m 60`.

To change the **account_name** associated with the interactive session, users can use the `-A` command line option. This option is useful for when a user has multiple allocations they belong to. For example, if I have allocations on accounts `TACC` and `Training`, I can use `-A` to set the allocation I want to be used like so: `-A TACC` or `-A Training`.

To change the **queue** to be different than the default `development` queue, users can use the `-p` command line option. For example, if a user wants to launch an interactive session on one of Stampede2's `Skylake` (SKX) compute nodes, they would use the command line option `-p skx-dev` or `-p skx-normal`. Note that the `-p skx-large` queue will be out of scope for most project users. You can learn more about the different queues of Stampede2 here.

Note: For the scope of this section, we will be using the default `development` queue.

To start a thirty-minute interactive session on a compute node in the development queue with our `TRAINING-HPC` allocation:

```
login1$ idev -A TRAINING-HPC
```

If launch is successful, you will see output that includes the following excerpts:

```
...
-----------------------------------------------------------------
      Welcome to the Stampede2 Supercomputer
-----------------------------------------------------------------
...

-> After your idev job begins to run, a command prompt will appear,
-> and you can begin your interactive development session.
-> We will report the job status every 4 seconds: (PD=pending, R=running).

-> job status:  PD
-> job status:  R

-> Job is now running on masternode= c449-0015...OK
...
c449-0015[knl](268)$
```

Exercise

Let's revisit the job we ran in the previous section. This time, we will be going through each command we entered into `job.slurm` interactively.

```
c449-0015[knl](268)$ pwd
/home1/03439/wallen/IntroToLinuxHPC/Lab04
c449-0015[knl](269)$ ls
data job.slurm results vina_job.o864828
```

```
c449-0015[knl](270)$ echo "starting at:"
starting at:
c449-0015[knl](271)$ date
Mon Jun 29 0X:XX:XX CDT 2020
c449-0015[knl](272)$ module list

Currently Loaded Modules:
#  it is okay if you have loaded modules from past sessions

c449-0015[knl](273)$ module load intel/17.0.4
c449-0115[knl](274)$ module load boost/1.64
c449-0115[knl](275)$ module load autodock_vina/1.1.2
c449-0115[knl](276)$ module list

Currently Loaded Modules:
1) intel/17.0.4
2) boost/1.64
3) autodock_vina/1.1.2     #the order in which the modules are listed does not matter

c449-0015[knl](277)$ cd data/
c449-0015[knl](278)$ vina --config configuration_file.txt --out ../results/output_
↪ligands.pdbqt
###############################################################
# If you used AutoDock Vina in your work, please cite:        #
#                                                             #
# O. Trott, A. J. Olson,                                      #
# AutoDock Vina: improving the speed and accuracy of docking  #
# with a new scoring function, efficient optimization and     #
# multithreading, Journal of Computational Chemistry 31 (2010)  #
# 455-461                                                     #
#                                                             #
# DOI 10.1002/jcc.21334                                       #
#                                                             #
# Please see http://vina.scripps.edu for more information.    #
###############################################################

Detected 272 CPUs
WARNING: at low exhaustiveness, it may be impossible to utilize all CPUs
Reading input ... done.
Setting up the scoring function ... done.
Analyzing the binding site ... done.
Using random seed: -31156704
Performing search ...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************
done.
Refining results ... done.

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1        -12.3      0.000      0.000
   2        -11.1      1.223      1.866
   3        -11.0      3.000     12.459
   4        -10.5      2.268     12.434
   5        -10.4      2.272     13.237
```

(continues on next page)

(continued from previous page)

```
   6         -10.3       3.146      13.666
   7         -10.3       3.553      12.345
   8         -10.2       1.827      13.667
   9          -9.8       2.608      12.630
Writing output ... done.

c449-0015[knl](279)$ echo "ending at:"
c449-0015[knl](280)$ date
Mon Jun 29 0X:XX:XX CDT 2020
```

To exit an interactive session, you can either use `logout` or wait until the connection to the compute node is closed by the remote host.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search