Command-Line Bioinformatics Primer

Prepared for BIOL2010 by Andrew Leith
Presented by Nicola Neretti

Part I: Introduction to Linux

The Command Line

The original way to interact with a computer, dating back to the 1960s

Why use it in this day and age?

- Greater flexibility compared to a graphical user interface (GUI)
- CLI utilities are easy to automate

Command Line Overview

There are several important aspects of using the CLI

01	**Navigation**	• pwd (print working directory) • ls (list files) • cd (change directory)
02	**File Permissions**	• chmod (change permissions) • chown (change owner) • chgrp (change group)
03	**File Manipulation**	• mkdir (make directory) • mv (move file(s)) • cp (copy file(s))
04	**Text Manipulation**	• cat (show file contents) • grep (search for text) • cut (extract a column)
05	**Miscellaneous**	• Downloading files • Function manuals • Resolving errors

CLI Navigation: `pwd` and `cd`

`pwd`

- Stands for ‘print working directory’
- Displays the full file path of the directory (folder) you’re currently in

```
[aleith@login002 ~]$ pwd
/users/aleith
[aleith@login002 ~]$ ls
4-1
5b
anaconda
```

`cd`

- Stands for ‘change directory’
- Allows to move around from one folder to another
    - Usually if you type part of a location and hit the TAB key, it will try to autocomplete

```
[aleith@login002 ~]$ pwd
/users/aleith
[aleith@login002 ~]$ cd anaconda
[aleith@login002 anaconda]$ pwd
/users/aleith/anaconda
```

CLI Navigation: `ls`

`ls`

- stands for ‘list’
- lists the contents of the current working directory
- Can be colored to depict contents

```
[aleith@login002 ~]$ ls
4-1                     data
5b                      Desktop
anaconda                Downloads
batch.script            Figure2a.eps
batch_scripts           Figure2b.eps
```

Detour: Command Options

Many Linux commands have options that change their behavior

Most options are set using a minus (-) and a letter

`ls` example

- by default, `ls` only shows non-hidden files
- To show all files, you would use:

```
ls -a
```

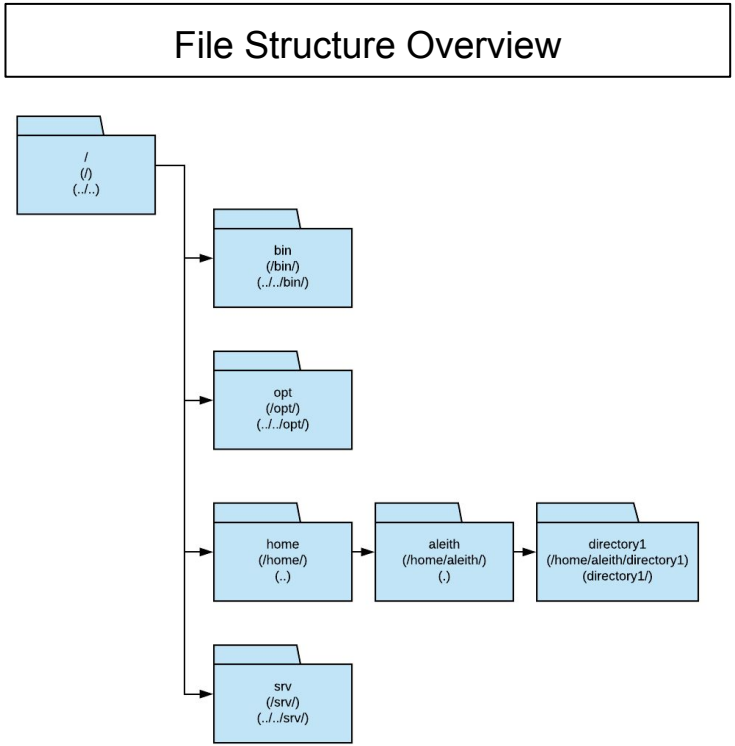
```
[aleith@login002 ~]$ ls -a
.                       ._.DS_Store
..                      .DS_Store
4-1                     .esd_auth
5b                      .felix
anaconda                Figure2a.eps
.bash_history           Figure2b.eps
```

CLI Navigation: File Paths

File path shows the location of a file or directory

Two fundamental ways to refer to file path on Linux:

- Relative file path:
  - the location of a file with respect to your current working directory (i.e the location you are in)
- Absolute file path:
  - sometimes described as a ‘full file path’, is that file’s true location on the computer and is invariant



CLI Navigation: File Paths Notation

Several pieces of shorthand notation often used in defining a relative file path

- `.` stands for the current working directory (i.e. the output of `pwd`)

  Put in a figure for each bullet showing how it's written on the command line

- `..` stands for the parent directory, i.e. the directory that contains the current working directory

- `~` ('tilde') stands for the current user's (i.e. your) home directory

An absolute file path is the path with respect to the root directory ('/')

Detour: Relative vs Absolute File Paths

The following two paths are very different:

`anaconda/bin/` is the ‘bin’ directory inside the ‘anaconda’ directory inside the **current** directory

Same as `./anaconda/bin/`

`/anaconda/bin/` is the ‘bin’ directory inside the ‘anaconda’ directory inside the **root** directory

To avoid ambiguity, it is generally a good idea to stick to absolute file paths whenever possible

File Permissions

- Interacting with files on Linux requires ‘permission’
- Three fundamental permissions: `read(r),write(w)and execute(x)`
- There are also three *kinds* of permissions:
  - Permissions for the user (u)
  - Permissions for the user’s group (g)
  - Permissions for everyone else (o) on the system
- To view a file’s permissions on Linux, use `ls -lh`

```
[aleith@login002 linux_tutorial]$ ls -lh
total 0
-rw-r--r-- 1 aleith cbc 0 Mar 15 16:18 randomfile.txt
```

File Permissions - Explained

```
[aleith@login002 linux_tutorial]$ ls -lh
total 0
-rw-r--r-- 1 aleith cbc 0 Mar 15 16:18 randomfile.txt
```

`-rw-r--r--` (colorized for clarity) means ‘user has read and write permission’, ‘user’s group has read permission’, and ‘everyone else has read permission’

`aleith cbc` means this file is owned by user ‘aleith’ from group ‘cbc’

`0` refers to the file’s rounded file size

The last-modified date is also included for each file

File Permissions - Modifications

Change Permissions

- A file's permissions can be changed by the owner using `chmod`
  - Example:user 'aleith' gives himself execute permission '`randomfile.txt`' command

```
[aleith@login002 linux_tutorial]$ ls -lh
total 0
-rw-r--r-- 1 aleith cbc 0 Mar 15 16:18 randomfile.txt
[aleith@login002 linux_tutorial]$ chmod u+x randomfile.txt
[aleith@login002 linux_tutorial]$ ls -lh
total 0
-rwxr--r-- 1 aleith cbc 0 Mar 15 16:18 randomfile.txt
```

Change Ownership

- Transfer ownership of a file to the user specified as 'new owner'
  - `chown [new owner] [file]`
- Transfer the file to another group without changing the user
  - `chgrp [new group] [file]`

File Manipulation - Moving Files

- To create a new directory from the command line use
  - `mkdir [directory name]`
- Directories created in this manner cannot contain spaces

```
[aleith@login002 linux_tutorial]$ mkdir directory1
[aleith@login002 linux_tutorial]$ ls
directory1  randomfile.txt
```

- To move a file use the following command
  - `mv [target] [destination]`
  - The command moves ‘target’ to the directory specified as ‘destination’ or renames the file

```
[aleith@login002 linux_tutorial]$ ls
directory1  randomfile.txt
[aleith@login002 linux_tutorial]$ mv randomfile.txt newname.txt
[aleith@login002 linux_tutorial]$ ls
directory1  newname.txt
[aleith@login002 linux_tutorial]$ mv newname.txt directory1/
```

File Manipulation - Copying Files

Files can be copied by typing

```
cp [original filename] [new filename]
```

The ‘copy’ command simply duplicates a file in its entirety

```
[aleith@login002 directory1]$ ls
randomfile.txt
[aleith@login002 directory1]$ cp randomfile.txt copiedfile.txt
[aleith@login002 directory1]$ ls
copiedfile.txt  randomfile.txt
```

To copy an entire directory, you must use `cp -r` (‘recursive copy’)

Detour: Wildcard

- The `*` character is known as the ‘wildcard’, and it is used to mean ‘anything’
  - For example, `ls *.txt` will run ‘`ls`’ on every file that ends in ‘`.txt`’
- Two more examples.
  - list any files beginning with ‘random’

```
[aleith@login002 directory1]$ ls
copiedfile.txt  randomfile.txt  script.sh
[aleith@login002 directory1]$ ls *.txt
copiedfile.txt  randomfile.txt
```

```
[aleith@login002 directory1]$ ls random*
randomfile.txt
```

  - list any files that start with anything, have a ‘`d`’ in the middle, and then end with anything

```
[aleith@login002 directory1]$ ls *d*
copiedfile.txt  randomfile.txt
```

File Manipulation: Deleting Files

To delete a file, we use the following command

```
rm [filename]
```

The ‘remove’ command permanently deletes a file. Linux has no ‘Trash’ feature like Windows and OSX do - `rm` is in fact directly analogous to the ‘Empty Trash’ option on those systems

Text Manipulation: Viewing Text Files

To view the contents of a text file use

- The `less` command
  - it will display the first few lines of the file and allow you to scroll through it.
  - Pressing ‘q’ will close this browser
- The `cat` command
  - Prints out the complete file to screen
- The `head` command
  - Prints out first n lines to screen
- The `tail` command
  - Prints out last n lines to screen
- Text editors: `nano` [link to tutorial], `vim` [link to tutorial]

Text Manipulation: Redirecting Output

- By default, most commands (e.g. `echo`) will print their output to the screen
- The CLI has operators that change this behavior by *redirecting* that output
- The `> [file]` operator redirects output to '`file`', overwriting it if it already exists

```
[aleith@login002 directory1]$ echo "hello world"
hello world
[aleith@login002 directory1]$ echo "hello world, redirected" > hello.txt
[aleith@login002 directory1]$ ls
copiedfile.txt  hello.txt  randomfile.txt  script.sh
[aleith@login002 directory1]$ cat hello.txt
hello world, redirected
```

- The related `>> [file]` operator appends output to '`file`'

```
[aleith@login002 directory1]$ echo "hello world, appended" >> hello.txt
[aleith@login002 directory1]$ cat hello.txt
hello world, redirected
hello world, appended
```

- The `| [command]` operator, '`pipe`', redirects output directly to another command
  - We will see this in action later

Text manipulation: Searching Inside Files

- The `grep` command allows us to search through a file for any entries that match a specified pattern
  - `grep [pattern] [filename]`
- Example:
  - we can search our downloaded dictionary for all words containing the letter sequence ‘aar’

```
[aleith@login002 grep_examples]$ grep "aar" dictionary.txt
aardvark
aardwolf
bazaar
tumultuaary
zaar
```

Text manipulation: sort, cut, paste

To sort (by alphabetical order by default), type

```
sort [file]
```

To extract a column from a text file, type

```
cut -f[column number] [file]
```

e.g. `cut -f2 textfile.txt`

To combine files horizontally (analogous to cbind in R), type

```
paste [file 1] [file 2] ... [file n]
```

Text manipulation: Using Pipe

But how can we count the number of matches if there are hundreds of them?

- Combine `grep` and `wc -l` using a pipe
  - To know how many words in our dictionary contain the letter ‘a’ anywhere, we can use

```
[aleith@login002 grep_examples]$ grep "a" dictionary.txt | wc -l
40387
```

  - To find all words containing ‘the’ and counting the variations
  - `grep ‘the’ dictionary.txt | sort | uniq -c`
- Using the wrong operator ends up with weird behavior and a file named ‘wc’

```
[aleith@login002 grep_examples]$ grep "a" dictionary.txt > wc -l
[aleith@login002 grep_examples]$ ls
dictionary.txt  wc
```

Misc: Downloading Files

The simplest way to download a file on Linux is

```
wget [url]
```

- It downloads a file as-is, with the same filename, and places it in your current working directory.
- Can be used with most protocols including
  - FTP

[placeholder until we figure out what to do with the dictionary]

Misc: Interrupting Command Execution

Sometimes, it will be necessary to stop a command before it completes (e.g. upon making a typo with a wildcard and copying thousands of files instead of a few)

To interrupt a command, type CTRL+c (this combination is valid on both Windows and OSX - do not substitute COMMAND for CTRL on OSX or it will not work)

Misc: Function Manuals

One of the most important resources for understanding Linux commands is the manual associated with each command

- On most versions of Linux, you can access the manual for a function by

  ```
  man [function]
  ```

  - e.g. `man cp`
- For a short version use `[function] --help`
  - e.g. `cp --help`
- Note that this syntax will not necessarily completely translate to other UNIX-like operating systems such as OSX

Misc: Common Errors

‘command not found’

- This error means that the computer does not know how to process what you’ve typed
  - It could be due to a simple typo

```
[aleith@login001 ~]$ cpr --help
bash: cpr: command not found
```

  - valid command not in PATH

```
[aleith@login001 ~]$ samtools --help
bash: samtools: command not found
```

Detour: The PATH

To view your PATH, you can type

```
echo $PATH
```

Here, the ‘$’ basically means ‘bash variable’ - PATH is a globally-defined variable on every Linux system

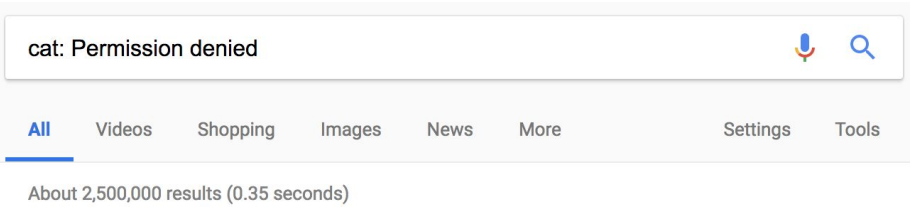
Brown’s compute cluster has functionality that will temporarily modify your path in response to something called a `module load` command, which we will cover in the second half of this talk

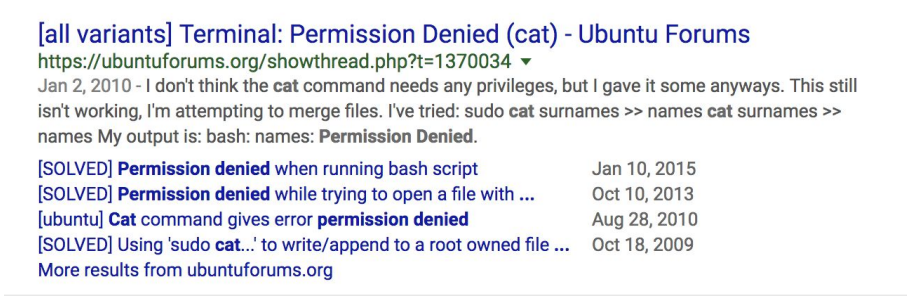
Misc: Common Errors and Resolving them

‘Permission denied’

- Occurs when you lack the permissions to perform a given operation
  - This error usually comes about when you have downloaded a file and it’s been created with weird permissions

To figure out the origin of and solution to an answer, Google the error - but remove the specific filenames and paths first or you won’t get sensible results





Part II: Brown’s Compute Cluster

Introduction to CCV

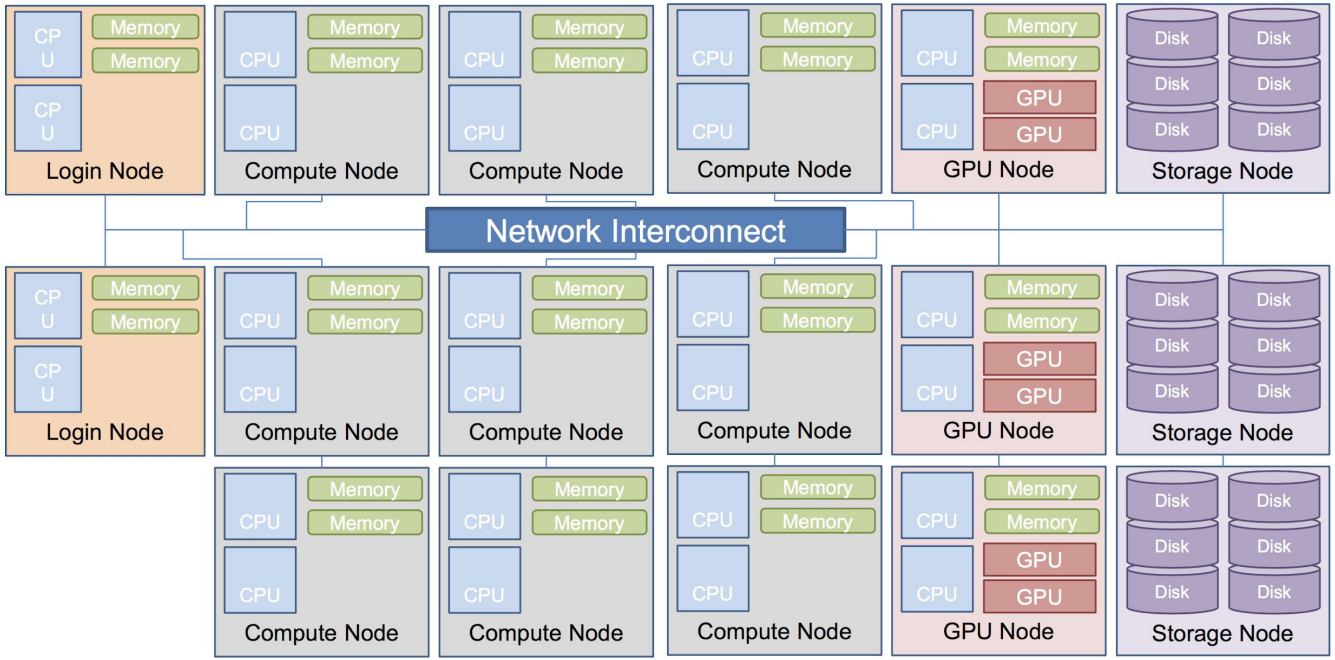
Brown has a compute cluster operated by CCV, the Center for Computation and Visualization

CCV offers services to researchers affiliated with Brown

Chief among their resources is ‘Oscar’, the compute cluster itself

What is a Compute Cluster?

Multiple interconnected computers that can be provisioned for running jobs



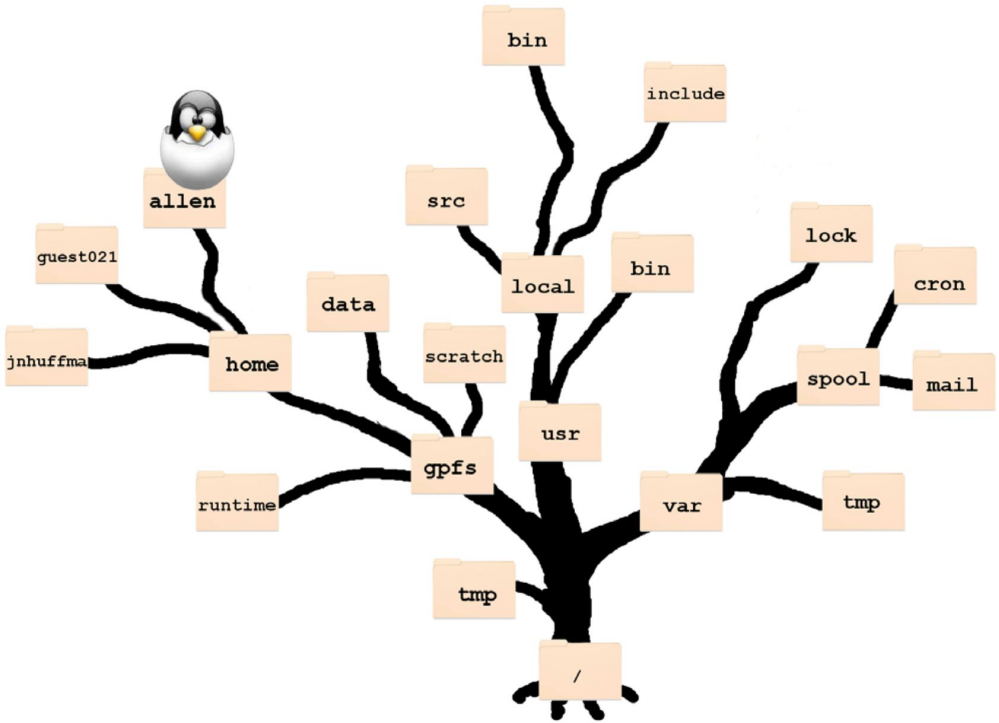
What are Oscar's Specifications?

8708 cores, 469 nodes

Node summary (February 2018)

Number of Nodes	Cores per node	Memory per node (Usable)	CPU Frequency	Processor
4	24	188 GB	2.6 GHz	Intel Xeon Gold 6126 (Skylake)
95	24	126 GB	2.2 GHz	Intel Xeon E5-2650 v4 (Broadwell)
1	24	110 GB	2.2 GHz	Intel Xeon E5-2650 v4 (Broadwell)
44	20	126 GB	2.3 GHz	Intel Xeon E5-2650 (Haswell)
144	20	126 GB	2.5 GHz	Intel Xeon E5-2670 v2 (Ivy Bridge)
151	16	63 GB	2.6GHz	Intel Xeon E5-2670 (Sandy Bridge)
2	16	126 GB	2.6GHz	Intel Xeon E5-2670 (Sandy Bridge)
1	16	55 GB	2.6GHz	Intel Xeon E5-2670 (Sandy Bridge)
16	12 8 x 2304 CUDA cores	128 GB 3 GB GDDR5	2.1 Ghz 863 Mhz	Intel Xeon E5-2620v2 (Ivy Bridge) NVIDIA GeForce GTX 780 (Kepler)
2	32 8 x 3584 CUDA cores	756 GB 11 GB GDDR5X	2.6 GHz 1582 MHz	Intel Xeon E5-2697A v4 (Broadwell) NVIDIA GeForce GTX 1080Ti
2	64	512 GB	2.6 Ghz	AMD Opteron 6282 SE (Interlagos)
4	64	512 GB	2.8 Ghz	AMD Opteron 6386 SE (Piledriver)

Oscar File System Overview



Connect to oscar and Run

Add commands here to

- Login
- Copy script
- Submit batch job

Connecting to Oscar via a Terminal

- To connect to Oscar, run the command
  - `ssh [user]@ssh.ccv.brown.edu`
- To use a graphical client use a slightly different command
  - `ssh -Y [user]@ssh.ccv.brown.edu`
- The ‘-Y’ option will enable graphics forwarding
  - Necessary to use any application with a GUI e.g. Firefox, RStudio, or the Matlab environment
- Without specifying ‘-Y’, attempts to display graphics will cause errors

```
[aleith@login001 directory1]$ firefox
Error: no display specified
```

Connecting to Oscar via a Terminal - Timeout

If left idle, ssh connections will timeout

Attempting to type into a terminal connected to a timed-out session will yield ~10 seconds of lag, and then an error message:

```
packet_write_wait: Connection to [ip] port 22: Broken pipe
```

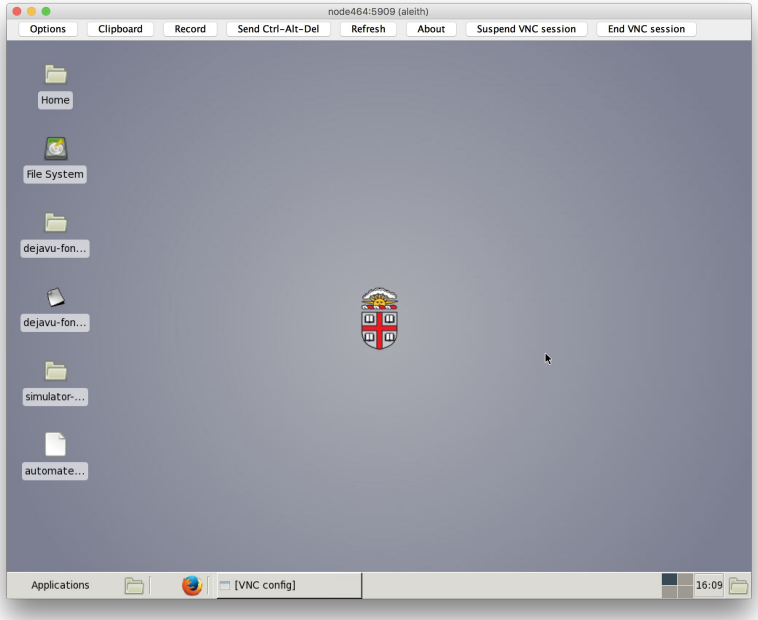
Batch jobs running in the background will not be affected by this disconnection, though any interactive jobs running in the foreground (e.g. copying files) will be

Connecting to Oscar Through Remote Desktop

CCV also provides a VNC client to connect to Oscar

https://web1.ccv.brown.edu/technologies/vnc

When logging into the VNC, you specify parameters such as the number of CPUS and memory



Getting Scripts onto Oscar

There are two primary ways to transfer files to and from Oscar

- SFTP (use a program to transfer them) or a command [link tutorial]

- Samba (mount Oscar like an external hard drive)
  - CCV's Samba tutorial can be found on their website
    - : https://web1.ccv.brown.edu/doc/cifs

Using a Compute Cluster

The question then becomes: how does one actually run anything?

Two Step process:

- User must create a special type of script called a ‘batch script’
- ‘Batch script’ is used for running so-called ‘batch jobs’
    - Specify one or multiple jobs

‘Batch script’ is submitted to the ‘resource manager’

    - A utility that allocates nodes based on availability and requirements

Oscar uses a resource manager called ‘Slurm’; it will put jobs in a queue based on the computational requirements requested and the user’s job priority

Batch Script - Formatting

All batch scripts require a header that follows a specified format - otherwise, Slurm will be unable to allocate resources to it

`#!/bin/bash`	The typical shebang statement
`#SBATCH --time=144:00:00`	The time to request resources for
`#SBATCH --cpus-per-task=8`	The CPUs to request
`#SBATCH  --mem=64G`	The memory to request

Batch Script - Formatting (continued)

`#SBATCH -J exac`	The name under which to list the job
`#SBATCH -o exac.out`	The file for ‘standard out’
`#SBATCH -e exac.err`	The file for ‘standard error’

Output is provisioned into two files, ‘standard out’ and ‘standard error’ (output into the former, and errors mostly into the latter but occasionally the former)

Other parameters can be found at the following address:

https://web1.ccv.brown.edu/doc/jobs

Batch Script - Submitting

To submit a batch script with use

- `sbatch [script name]`
- You will receive a message stating that your job has been submitted.

To monitor job status on OSCAR use

- `myq`
- Shows any pending or running jobs associated with your own user account
  - It is useful to check if a job has failed
  - if you submit an alignment you expect to take hours, and after 30 seconds no job are running, there is an error in your script

Interactive Jobs

The terminal you see when first sshing into Oscar is running on a ‘login node’, a shared machine with low capabilities

Computationally-intensive jobs run on the login node will immediately die

It is possible, however, to access a node that works like a fully-capable terminal

To access such a node, type

```
interact
```

Interactive Jobs

An example of an `interact` command is

```
interact -n 20 -t 01:00:00 -m 10g
```

`-n` refers to the number of cores (20)

`-t` refers to the time the session will last (1 hour)

`-m` refers to the memory per node (10 GB)

The full list of options can be found at the following link:

https://web1.ccv.brown.edu/doc/jobs

Modules

Multiple versions of the same software tool are provided on OSCAR

- To use a given tool, you must first ‘load’ that tool as a ‘module’ as follows
  - `module load [software name]`
- Example:
  - `module load samtools`

- To see a list of all software available on Oscar, use
  - `module avail`
- To see all of the modules currently loaded use
  - `module list`
- To unload a loaded module, type
  - `module unload [software name]`