#Command Line

reminder of some tools

- `wc` - count the number of lines,words,characters
- `tail` - see the last N lines of a file
- `head` - see the first N lines of a file
- `cat` - print out entire file to screen
- `sed` - 'stream editor' - edit data stream on the fly
- `curl` - downloading tool for web/ftp data streams
- `which` - list path to a program based on the $PATH
- `pwd` - print the current working directory
- `ps` - processes running on the system
- `man` - view manual pages about a command or program
- `date` - date and time
- `time` - prefix a command/program, report how long it took to run
- `find` - find files/folders by name or other property
- `du` - reports disk usage (e.g. how big a file or folder is)
- `awk` - a simple language for processing files/great for column delimited data

Reminder of some commands while at the command line

- `^` means "Control key"
- cancel a running application: `^C`
- end a session: `^D` (End of File message)
- Tab to try to autocomplete (applications, filenames, directories)
- While typing on cmdline - jump to end of line: `^E`
- Get back to the beginning of line: `^A`
- Up and down keys cycle through history of commands
- Type `!!` to execute the last command
- Type `history` to see list of previous commands
- Type `!NUMBER` to excute cmd from that list
- Type `!g` to run the last command that started with `g`

Processes

UNIX allows multiple processes (programs) to be run at the same time. While at the command line you can run a specific process, but your command line will be blocked until that program is finished.

Try this which will run a task which just pauses for 30 seconds.

```
sleep 30
```

Jobs are run in the foreground by default. While a job is running use `^Z` to suspend job.

```
sleep 30
^Z
[1]+  Stopped                 sleep 30
```

To keep the job running but put it in the **background** use the command `bg` which will puts process in background.

```
$ bg
[1]+ sleep 30 &
```

If you want to put the job back in the foreground so you can interact with it or force cancel it use the command `fg`.

```
$ fg
fg
sleep 30
```

To launch a job directly into the background put an `&` at the end.

```
$ sleep 30 &
```

It can still be brought to the foreground with `fg`.

A typical use if you are using a tool which generates graphical interface that you will interact with is to launch it in background. It will print out the process id of the command that is running.

```
$ emacs &
[1] 25341
```

More file manipulation

Copying files

To copy a file use the command `cp`.

```
$ cp one.txt two.txt# copy one file to another
$ mkdir books # make a directory
$ cp one.txt books        # copy into a directory
$ ls books                # list the contents
one.txt
$ cp books more_books     # copy the folder, will fail
cp: books is a directory (not copied).
$ cp -r books more_books  # recursive copy succeeds
$ cp one.txt two.txt books # can copy more than one at a time
                           # will also OVERWRITE the previous
```

```
                              # one.txt that was in the folder
$ ls books
$ ls more_books
```

The command `rsync` can also be used to copy files between folders or between computers.

Here we copy a file that is located on your laptop called LOCALFILE onto the HPCC and will put it in the folder bigdata which is located in your home directory.

```
[your laptop] $ rsync -a --progress LOCALFILE USER@cluster.hpcc.ucr.edu:bigdata/
```

Can also specify an explicit path (starts with `/`).

```
[your laptop] $ rsync -a --progress LOCALFILE USER@cluster.hpcc.ucr.edu:/bigdata/gen220/USE
```

Can copy FROM HPCC to your local computer

```
[your laptop] $ rsync -a --progress USER@cluster.hpcc.ucr.edu:/bigdata/gen220/share/simple/y
```

Moving files

Moving files is just renaming them.

```
$ mv one.txt three.txt     # rename one.txt to three.txt
$ mv three.txt books       # relocate three.txt to books folder
$ cd books
$ mv one.txt two.txt three.txt .. # move these files back UP one
directory
$ ls                       # nothing in the 'books' directory
$ cd ..                    # go back
$ ls                       # these files are in the current folder
one.txt two.txt three.txt books more_books
$ ls books                 # is now empty, we moved everything
                           # out of there
```

Running programs

How does UNIX determine what program to run?

Try typing `echo $PATH` to see your search directory. Also do `env` to see all environment variables.

You can use the command `which` to tell you where a program is located

```
which nano # will tell you where the nano program
           # is located
```

On the UCR HPCC there are many installed applications through a UNIX module system. To load a module means to make that program part of your path and in some cases will set other environment variables.

For example to get access to the BLAST suite.

```
$ which blastn
/usr/bin/which: no blastn in ....

$ module load ncbi-blast
$ which blastn
/opt/linux/centos/7.x/x86_64/pkgs/ncbi-blast/2.2.30+/bin/blastn
```

There are multiple versions installed on the system

```
module avail ncbi-blast

---- /opt/linux/centos/7.x/x86_64/modules ----------------------------
ncbi-blast/2.2.22+          ncbi-blast/2.2.30+(default) ncbi-blast/2.6.0+
ncbi-blast/2.2.25+          ncbi-blast/2.2.31+          ncbi-blast/2.7.1+
ncbi-blast/2.2.26           ncbi-blast/2.3.0+           ncbi-blast/2.8.0+
ncbi-blast/2.2.26+          ncbi-blast/2.4.0+           ncbi-blast/2.8.1+
ncbi-blast/2.2.29+          ncbi-blast/2.5.0+           ncbi-blast/2.9.0+
```

You can load a specific version

```
module load ncbi-blast/2.9.0+
which blastn
/opt/linux/centos/7.x/x86_64/pkgs/ncbi-blast/2.9.0+/bin/blastn
module unload ncbi-blast
/usr/bin/which: no blastn in ....
```

See what versions of modules you currently have loaded

```
module list
Currently Loaded Modulefiles:
  1) slurm/19.05.0                 4) texlive/2017                  7) geos/3.7.1
  2) openmpi/4.0.1-slurm-19.05.0   5) pandoc/2.0                    8) gdal/2.1.3
  3) ggobi/2.1.11                  6) netcdf/4.4.1.1                9) hdf5/1.10.1
```

Running programs

`curl` is useful downloading from remote sites. URLs either FTP, HTTP, or HTTPS.

```
 $ curl http://www.uniprot.org/uniprot/E3Q6S8.fasta
 >tr|E3Q6S8|E3Q6S8_COLGM RNAse P Rpr2/Rpp21/SNM1 subunit domain-containing protein
OS=Colletotrichum graminicola (strain M1.001 / M2 / FGSC 10212) GN=GLRG_02386 PE=4 SV=1
MAKPKSESLPNRHAYTRVSYLHQAAAYLATVQSPTSDSTTNSSQPGHAPHAVDHERCLET
```

```
NETVARRFVSDIRAVSLKAQIRPSPSLKQMMCKYCDSLLVEGKTCSTTVENASKGGKKPW
ADVMVTKCKTCGNVKRFPVSAPRQKRRPFREQKAVEGQDTTPAVSEMSTGAD
$ curl -OL http://www.uniprot.org/uniprot/E3Q6S8.fasta

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   345  100   345    0     0    724      0 --:--:-- --:--:-- --:--:--   724

$ curl -o myseqfile.fa  http://www.uniprot.org/uniprot/E3Q6S8.fasta
```

Often use the '-L' in curl to allow URL redirects. There are also resuming options and ways to pass in username/password for authenticated sites. Also

- wget - also gets web/FTP on commandline
- ncftpget - for ftp
- lftp - a command line FTP client - also works for http/web

Redirect output and input

> - write out the output to file (create it if empty, and overwrite if exists)

```
$ curl http://www.uniprot.org/uniprot/E3Q6S8.fasta > E3Q6S8.fa
```

>> - write out output to a file (create it if empty) but append to the end of the file

```
$ echo "my name is " >> what_is_my name
$ echo "Joe" >> what_is_my_name
$ cat what_is_my_name
my name is
Joe
```

< - This is for redirecting INPUT from a file. We'll talk about this more but is how we might pull a set of commands into a program expecting input

```
$ R --no-save < My_R_commands.R
```

Compression

File compression can save disk space, reduce file transfer time when copying between computers

- `gzip` for GNUzip compression. Single file at a time.
- `pigz` is parallelized and can use multiple processors

```
$ module load pigz
$ pigz file.txt
```

```
$ du -h data/Nc20H.expr.tab        # report how big the file is
656K    data/Nc20H.expr.tab
$ pigz data/Nc20H.expr.tab         # to compress
$ du -h data/Nc20H.expr.tab.gz     # report size of compressed file
236K    data/Nc20H.expr.tab.gz
$ pigz -d data/Nc20H.expr.tab.gz    # to uncompress
```

- `bzip2` for Bzip compression. Better compression than gzip but slower.
- pbzip2 is parallelized and can use multiple processors

```
$ bzip2 data/Nc20H.expr.tab        # compress with bzip2
$ du -h data/Nc20H.expr.tab.bz2    # report size of bzipped file
204K    data/Nc20H.expr.tab.bz2
$ bunzip2 data/Nc20H.expr.tab.bz2
```

`zcat`, `zmore` and `bzcat`, `bzmore` to read compressed files on the fly

Disk space usage of files

`du` - disk usage * -h – show result in human readable output (eg butes, Kilobytes, Gigabytes) $ –time - show time of last update

```
$ du -h /bigdata/gen220/shared/data_files/S_cerevisiae_ORFs.fasta
11M /bigdata/gen220/shared/data_files/S_cerevisiae_ORFs.fasta
$ du /bigdata/gen220/shared/data_files/S_cerevisiae_ORFs.fasta
10752   /bigdata/gen220/shared/data_files/S_cerevisiae_ORFs.fasta
$ du -h --time /bigdata/gen220/shared/data_files/S_cerevisiae_ORFs.fasta
```

Can also be used on folders to summarize the total size of contents of a folder.

Running programs

The pipe operator `|` allows you to instead of redirecting output to a file, redirect it to another program. Specifically the STDIN of the other program. This is very powerful and allows you to chain together different processes

```
$ zcat data/Nc20H.expr.tab.gz | wc -l
# output from zcat is printed to STDOUT and that is redirected
# to the command wc with the -l option which in turn
# expects input on STDIN.
# Output from a program can be fed to a pager like less
$ blastn -help | less
$ fasta36 query db | more
$ fasta36 query db | tee report.out | more
```

`tee` is a program which reads from STDIN and writes this BOTH to a file and to STDOUT. A way to monitor a program but to also detach from reading the messages and still capture it all to a file.

Multiple pipes can be used, and building together we can start to construct a series of queries. Will go into this more in detail in the next lecture but here we process a file and capture

```
# shows top ten results from a blast report
$ zcat data/blast.out.gz | head -n 10

# returns the total number of unique items found in column 1
$ zcat data/blast.out.gz | awk '{print $1}' | sort | uniq | wc -l

# take output from blast program, compress it on the fly to a new file
$ blastn -query query.fa -db db.fa -outfmt 6 | gzip -c > blastresult.gz
```

Using the HPCC cluster

On Biocluster there are a couple of folder structures to understand

- `/rhome/USERNAME` your home directory - limited space (20gb)
- `/bigdata/labname/USERNAME` your 'bigdata' folder (bigger space (100gb+)
- `/bigdata/labname/shared` shared folder space for your lab

Currently everyone is in the the gen220 'lab' during this course so you have access to `/bigdata/gen220/shared` and `/bigdata/gen220/USERNAME`

How much data am I using currently: https://dashboard.hpcc.ucr.edu

/scratch - local space on a cluster node which is FAST disk access but temporary (30 days)

Transferring data

Graphical tools: Filezilla - https://filezilla-project.org/download.php

Command-line:

```
# interactive FTP client
$ sftp USERNAME@cluster.hpcc.ucr.edu
# copy a file
$ scp USERNAME@cluster.hpcc.ucr.edu:fileoncluster.txt ./file-on-your-machine.txt
# copy a folder, recursively
$ scp -r USERNAME@cluster.hpcc.ucr.edu:/bigdata/gen220/shared/simple .
# rsync copies, but can check and only copy changed files
$ rsync -a --progress USERNAME@cluster.hpcc.ucr.edu:/bigdata/gen220/shared/simple .
```

```
# copy FROM your computer TO the cluster, swap order - here
# copy a folder back to your HOME directory
$ scp -r simple USERNAME@cluster.hpcc.ucr.edu:
```

Submitting jobs

Currently only shown login to the main "head" node (cluster.hpcc.ucr.edu)

To use the 6500 CPUs we need to submit job for running. This is called a job management or queueing system.

We use SLURM on the UCR system currently.

We use the SLURM queuing systems on HPCC. Read info here for more resources.
http://hpcc.ucr.edu/manuals_linux-cluster_jobs.html

Getting an interactive shell (eg get your own CPU to do work on)

```
$ srun --pty bash -l
$ srun --nodes 1 --ntasks 2 --mem 8gb --time 8:00:00 --pty bash -l
```

now you can type this in on the cmdline:

```
module load ncbi-blast
module load db-ncbi
curl https://www.uniprot.org/uniprot/Q5T6X5.fasta > Q5T6X5.fasta
blastp -num_threads 2 -query Q5T6X5.fasta  -db swissprot -out result.blastp
```

Batch/non-interactive job

You can also make this a job script (call it job.sh)

```
#!/bin/bash
module load ncbi-blast
module load db-ncbi
curl https://www.uniprot.org/uniprot/Q5T6X5.fasta > Q5T6X5.fasta
blastp -num_threads 2 -query Q5T6X5.fasta  -db swissprot -out result.blastp
```

Submit it with the following options

```
$ sbatch -N 1 -n 2 -p short job.sh
```

Requesting job resources

- number of CPUs: –ntasks N OR -n
- memory: –mem Xgb
- runtime: –time 12:00:00
- outputfile: –out results.log

Can also set these INSIDE the script

```
#!/bin/bash
#SBATCH --nodes 1 --ntasks 2 --mem 2gb --time 2:00:00
module load ncbi-blast
module load db-ncbi
curl https://www.uniprot.org/uniprot/Q5T6X5.fasta > Q5T6X5.fasta
blastp -num_threads 2 -query Q5T6X5.fasta  -db swissprot -out result.blastp
```