

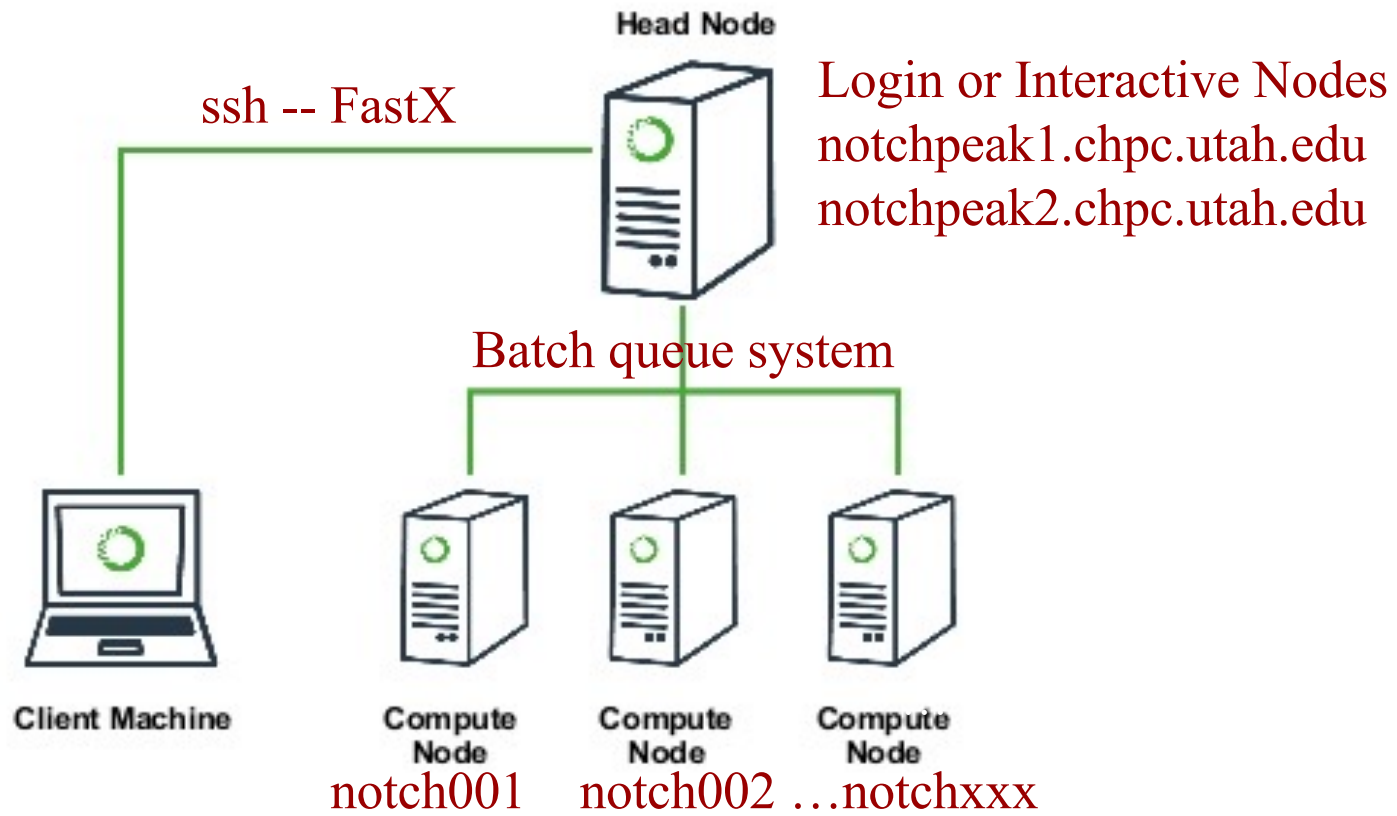
# Introduction to Linux – Part 1

Zhiyu (Drew) Li & Ashley Dederich  
Research Consulting & Faculty Engagement  
Center for High Performance Computing  
{zhiyu.li; ashley.dederich}@utah.edu

# Intro to Linux Series

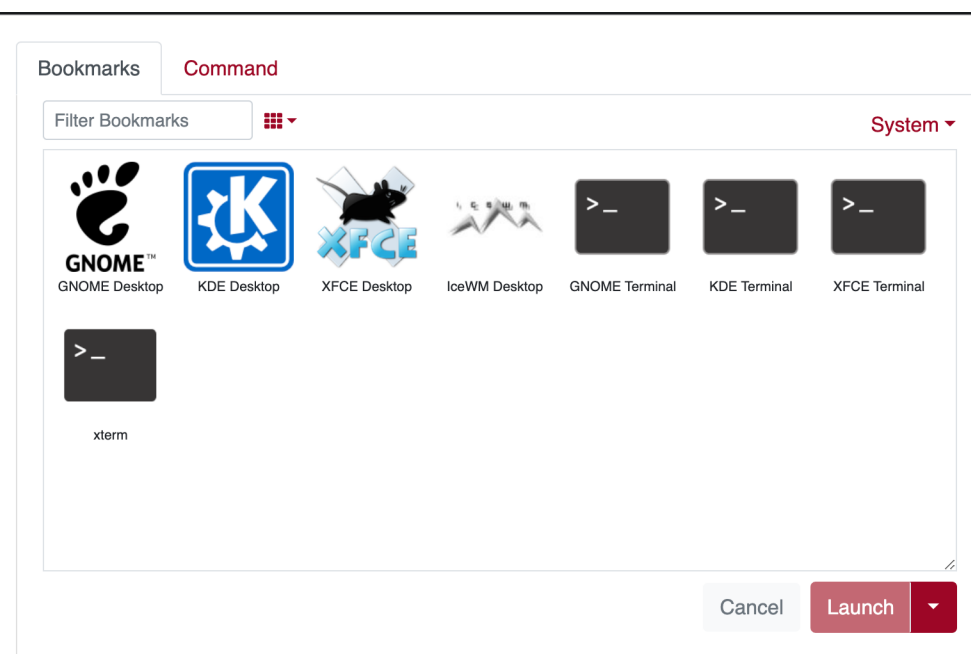
- Part 1 -- the ways to connect to CHPC Linux systems and go over some basic and important Linux commands.
- Part 2 -- introduce text editors, and then work on creating simple shell scripts.
- Part 3 -- cover more scripting techniques
- Part 4 (**Advanced; Fall semester only**) -- focus on software compilation
- Schedule and Slides are available at [www.chpc.utah.edu](http://www.chpc.utah.edu)

# Cluster Architecture Diagram



# Getting Started – Login with FastX


- Open web browser to:  
<https://linuxclass.chpc.utah.edu:3300>
- Enter temporary login and password (or your uNID & password if you have a CHPC account) and hit “SSH Login” button
- Click on the “Plus Icon” → select “XFCE Terminal” → Launch

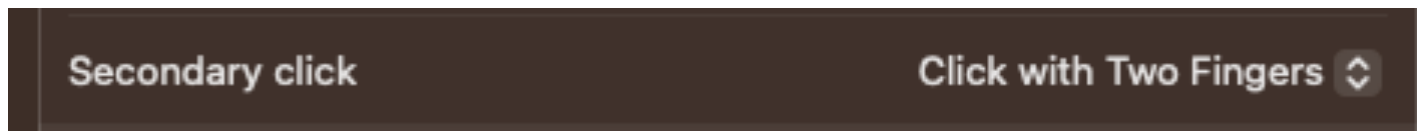


# FastX

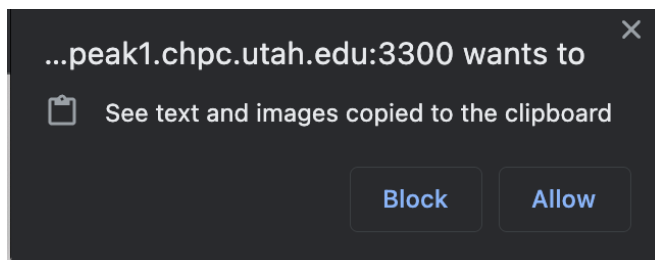
- <https://www.chpc.utah.edu/documentation/software/fastx.php>
- Remote graphical sessions in much more efficient and effective way than simple X forwarding
- Persistence - can be disconnected from without closing the session, allowing users to resume their sessions from other devices.
- Licensed by CHPC
- Desktop clients exist for windows, mac, and linux
- Web based client option
- Supported on all CHPC interactive nodes and the frisco nodes.

# Tips for Mac Users – Copy and Paste in web-based FastX terminal

- “Copy and Paste” only works in Google Chrome 
  - not working in Safari or Firefox so far
- Enable “Right-Click” on your Mac:
  - Apple Icon (upper-left) → System Settings → Search for “Trackpad” → Secondary click – “Click with Two Fingers”



- Perform a Copy and Paste:
  - Mouse select (highlight) command/text → “Right-click” (two-finger click) → Copy → Click on Terminal → “Right-click” → Paste



If Chrome prompts, click on Allow

# Windows – alternatives to FastX

- Need ssh client
  - PuTTY
    - <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
  - XShell
    - [http://www.netsarang.com/download/down\\_xsh.html](http://www.netsarang.com/download/down_xsh.html)
- For X applications also need X-forwarding tool
  - Xming (use Mesa version as needed for some apps)
    - <http://www.straightrunning.com/XmingNotes/>
  - Make sure X forwarding enabled in your ssh client
- OnDemand web portal
  - <https://ondemand.chpc.utah.edu>

# Alternatives to FastX on Mac/Linux

- Just open a terminal and execute:
- `ssh your_login@linuxclass.chpc.utah.edu`
- When running applications with graphical interfaces use `ssh -X` to enable X forwarding, for example:
- `ssh -X your\_unid@linuxclass.chpc.utah.edu`
  - (MacOS needs XQuartz <https://www.xquartz.org/>)
- OnDemand web portal
  - <https://ondemand.chpc.utah.edu>



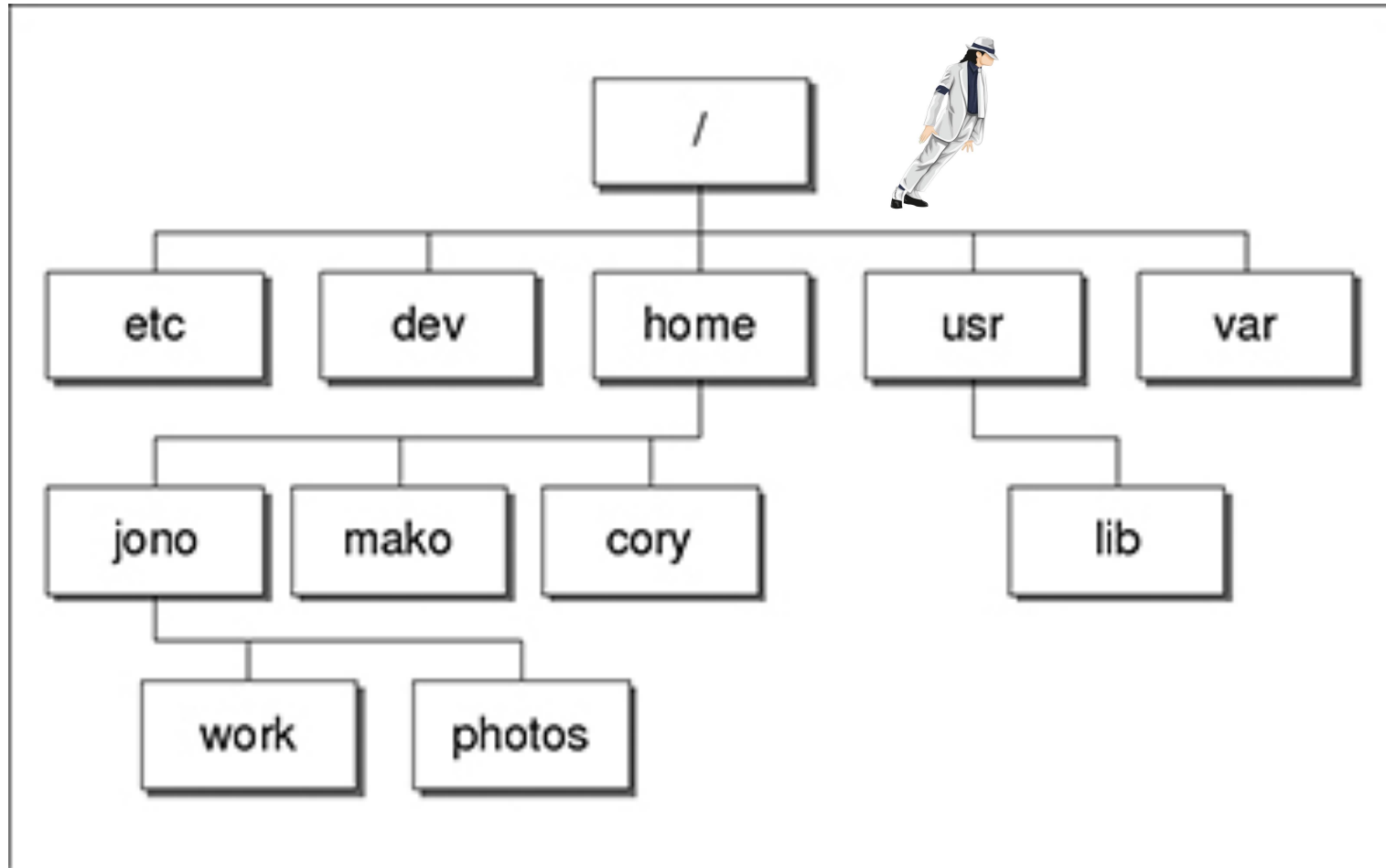


# Linux File System Directory Structure

- The file system is a tree directory structure
- levels are separated by / (forward slash) -
  - Note – not the \ (backslash) used in Windows!
- / --- refers to the “root” directory – the top-level directory that contains all other directories
- The Home directory is used to refer to a user’s base directory – this is where you will be upon login
  - On linuxclass server it will be /home/<yourusername>
  - CHPC clusters, this is in /uufs/chpc.utah.edu/common/home/<yourusername>
- /path/from/root → absolute path – has leading /
- path/without/leading/slash → relative path from current location
- . → current directory
- .. → parent directory (up one level)



# Linux Directory Structure



At CHPC cluster --- instead of **/home** we have **/uufs/chpc.utah.edu/common/home** under which we have all user directories

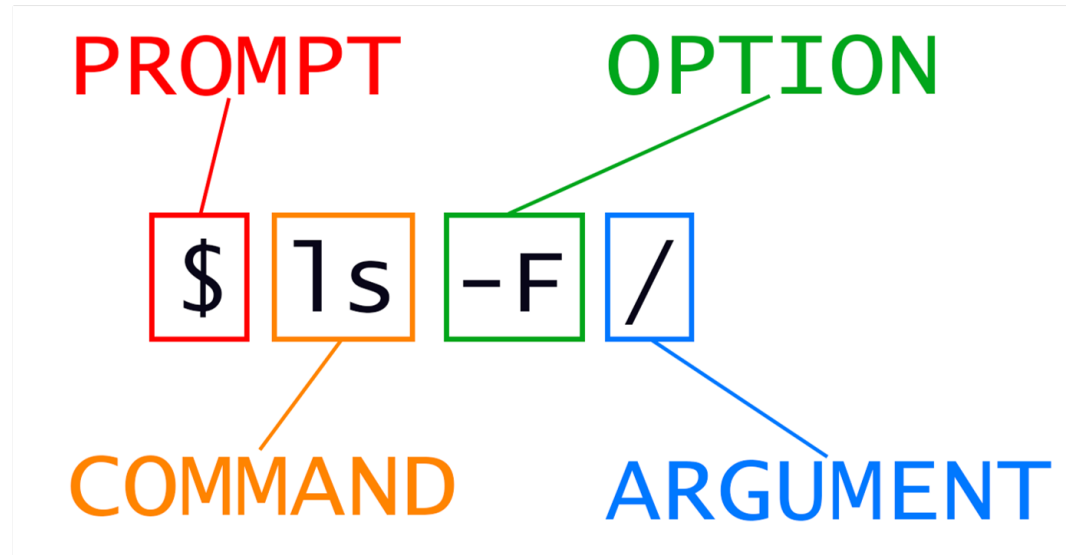
```
override@Atul-HP: ~  
total 212  
drwxrwxr-x 5 override override 4096 May 19 03:45 acadenv  
drwxrwxr-x 4 override override 4096 May 27 18:20 acadview_demo  
drwxrwxr-x 12 override override 4096 May 3 15:14 anaconda3  
drwxr-xr-x 6 override override 4096 May 31 16:49 Desktop  
drwxr-xr-x 2 override override 4096 Oct 21 2016 Documents  
drwxr-xr-x 7 override override 40960 Jun 1 13:09 Downloads  
drwxr-xr-x 1 override override 8980 Aug 8 2016 examples_desktop  
drwxr-xr-x 1 override override 45005 May 28 01:40 hs_err_pid1971.log  
drwxr-xr-x 1 override override 45147 Jun 1 03:24 hs_err_pid2006.log  
drwxr-xr-x 2 override override 4096 Mar 2 18:22 Music  
drwxrwxr-x 21 override override 4096 Dec 25 00:13 Mydata  
drwxrwxr-x 2 override override 4096 Sep 20 2016 newbin  
drwxrwxr-x 5 override override 4096 Dec 20 22:44 nltk_data  
drwxr-xr-x 4 override override 4096 May 31 20:40 Pictures  
drwxr-xr-x 2 override override 4096 Aug 8 2016 Public  
drwxrwxr-x 2 override override 4096 May 31 19:49 scripts  
drwxr-xr-x 2 override override 4096 Aug 8 2016 Templates  
drwxrwxr-x 2 override override 4096 Feb 14 11:22 test  
drwxr-xr-x 2 override override 4096 Mar 11 13:27 Videos  
drwxrwxr-x 2 override override 4096 Sep 1 2016 xdm-helper  
override@Atul-HP: ~$
```

# Shell Basics



- A Shell is a program that is the interface between you and the operating system (OS – e.g, linux)
- Command line interface – CLI – versus a GUI – or a graphical user interface
- Type commands on command line, send command by pressing enter, then the computer reads and executes the command and returns the results (NOTE – not all commands have output!)
- When commands are done they return to the PROMPT (more on prompts later)
- Commands can take flags/options that modify their behaviour
  - flags are formed with – (dash) and letter (sometimes --)
- Commands can also sometimes require an argument – this defines the item upon which the command acts

# General Syntax of Shell Commands



# Additional Shell Basics

- Linux is case sensitive!
- CHPC offers two basic shells - slightly different command syntax
  - `cshtcsh`
  - `shbash` (Bourne, Bourne again)
- While many shell commands are the same between shell types – there are syntax and behaviour differences
- Your account comes with a script that is executed upon login that sets a basic environment for your shell
- To check which shell you are using: `echo $SHELL`
  - Note `$SHELL` is an environment variable – more on these later
- To change shell for the session - enter name of shell you want at the prompt and hit enter
- For this class – we will be using bash

# Login & Prompts

- When you first login you will see a prompt (the prompt is set by the login script)
  - [u0028729@linuxclass:~]\$
  - [classXX@linuxclass:~]\$
- When you first login, you will be in your home directory
- To see your username: **whoami**
- To see your current directory: **pwd**
- Shortcuts
  - ~ (**tilde**) → your home directory
  - **\$HOME** → your home directory

# Exercise

- Log into linuxclass
- What is your shell?
- What is your username?
- What is the path of your current directory?

# Other Useful Items

- Up/down arrows go through past commands
- **history** – provides list of all recent commands; can ! followed by number from history list will put that command at the prompt
- Tab completion – of commands, paths, filenames – very useful
- Can edit previous commands – up and down arrow to get to command; then right, left arrow then delete any characters and type in new at cursor; cntrl-a gets to front of command line, cntrl-e to end of command line



# Basic Directory Commands

- **ls** – list contents of a directory
  - Flags to change output To see all flags
    - `ls --help`
    - `man ls`
- **cd** – move to directory (`cd test`)
  - **cd** without an argument moves you back to your home directory
  - `cd ..` -- moves you up one level
- **mkdir** – make directory (`mkdir test`)
  - Look at flags for **mkdir**
- **rmdir** – remove directory (`rmdir test`) – more on this later

# More on ls flags

- ❑ -l : long
- ❑ -a : All (including hidden files, also called dot files)
- ❑ -r : Reverse ordering while sorting
- ❑ -t : Timestamp

# Additional Commands

- ❑ **pushd** – change to directory and save previous location
- ❑ **popd** – change back to saved location
- ❑ **dirs** – show saved directories

# A Typical Problem: Nelle's Pipeline

- This example is from <https://swcarpentry.github.io/shell-novice/01-intro/index.html>
- Nelle Nemo, a marine biologist, has just returned from a six-month survey of the [North Pacific Gyre](#), where she has been sampling gelatinous marine life in the [Great Pacific Garbage Patch](#). She has 1520 samples that she's run through an assay machine to measure the relative abundance of 300 proteins. She needs to run these 1520 files through an imaginary program called `goostats.sh` she inherited. On top of this huge task, she has to write up results by the end of the month so her paper can appear in a special issue of Aquatic Goo Letters.
- if she has to run `goostats.sh` by hand using a GUI, she'll have to select and open a file 1520 times. If `goostats.sh` takes 30 seconds to run each file, the whole process will take more than 12 hours of Nelle's attention. With the shell, Nelle can instead assign her computer this mundane task while she focuses her attention on writing her paper.

# Exercise

- ❑ Make sure you are in your home directory
- ❑ Make a directory called **LinuxClass** and change into this directory
- ❑ Get the files
  - ❑ `wget https://home.chpc.utah.edu/~u0028729/CHPCPresentation/shell-lesson-data.zip`
  - ❑ `unzip shell-lesson-data.zip`
  - ❑ `cd shell-lesson-data`
- ❑ What directory are you in?
- ❑ List contents of this directory – see difference of a normal `ls`, `ls -l`, `ls -ltr`, and `ls -ltra`
- ❑ Change into the **exercise-data** directory
- ❑ List the contents
- ❑ Move back to the **shell-lesson-data** directory

# Exercise

- Start in the `shell-lesson-data` directory
- Make a new directory named `thesis`
- Re-list the contents of the `shell-lesson-data` directory
- Make new directories `project/data` and `project/results`
- List the contents of the `shell-lesson-data` directory
- List the contents of the `project` directory
- List the contents of the `north-pacific-gyre` directory
- Return to the `shell-lesson-data` directory

# Files & Filenames

- Directories can contain files and other directories
- Filenames are often in the format of name.extension
- Files that start with a “.” are hidden or “dot” files
- Extensions are useful for telling you what type of file it is – IF you follow the conventions (txt, pdf, jpg, etc)
  - The extensions also are used by the OS
  - The `file` command will tell you the file type
- Being careful with filenames can make your life easier – some guidelines:
  - Avoid special characters in names as you will have to handle these differently: space, tab, /, \, \$, leading -

# Nelle's Pipeline: Organizing Files

- Nelle is ready to organize the files that the protein assay machine will create. First, she creates a directory called **north-pacific-gyre** (to remind herself where the data came from).
- Each of her physical samples is labelled according to her lab's convention with a unique ten-character ID, such as 'NENE01729A'. This ID is what she used in her collection log to record the location, time, depth, and other characteristics of the sample, so she decides to use it as part of each data file's name. Since the assay machine's output is plain text, she will call her files **NENE01729A.txt**, **NENE01812A.txt**, and so on. All 1520 files will go into the same directory.
- Note that you may also consider to use dates, such as creating a directory called **2012-07-03**, to segregate data for a project where data may be gathered at different times.



# File commands

- ❑ **cat** – display contents of file
- ❑ **more** – display contents of file with page breaks
  - ❑ next page with Space key
  - ❑ “q” to exit
  - ❑ can also look at **less**
- ❑ **head** – display top of file (default is 10 lines, change with `-n` followed by number)
- ❑ **tail** – display end of file (default is 10 lines, change with `-n` followed by number)
- ❑ **grep** – search for pattern in file (`grep "pattern" test1`)
- ❑ **vi** – edit file (more on this later)
- ❑ **nano** – another file editor (more on this later)

# Exercise

- Navigate into the `north-pacific-gyre` directory
- View the contents of `NENE01720A.txt` file using
  - `cat`
  - `more`
  - `less`
  - `head`
  - `tail`

# File commands

- **cp** – copies file to a new name (`cp file1 file2`)
  - **cp -r** will recursively copy entire directories of files
- **mv** – renames file to a new file (`mv old new`) or location
  - Works for files and directories
- **touch** – creates an empty file if file does not exist OR changes time stamp if it does (`touch file`)
- **rm** – deletes file (`rm file1`)
  - Note shells DO NOT have a trash bin; rm is final!

# Exercise

- Change to the new `thesis` directory
- List the contents of the `thesis` directory
- Create a new file `draft.txt` in this directory using the `touch` command (saving editors for next time)
- List the contents of the `thesis` directory
- Rename the file `draft.txt` to `empty.txt`
- You can also move a file to a different directory
  - Move `empty.txt` to the `shell-lesson-data` directory
- In the `shell-lesson-data/exercise-data` directory make a copy of the entire `writing` directory, naming it `writing-backup`
- Remove the `empty.txt` file
- Remove the `writing-backup` directory

# Wildcards

- multiple files can be specified via wildcards
- \* - matches any number of letters including none
- ? - matches any single character
- [ ] - encloses set of characters that can match the single given position
- - used within [ ] denotes range of characters

# Exercise

- Move to the `shell-lesson-data/exercise-data/proteins` directory
- Using the `ls` command, test the use of wildcards
  - `*.pdb`,
  - `p*.pdb`
  - `*ethane.pdb`
  - `?ethane.pdb`

# Command output redirection

- **>** redirect output to a file (instead of to screen)
  - will create file if it does not exist
  - will overwrite the previous contents if it does exist
  - **cat file1.dat > file4.dat**
- **>>** append to a file
  - **cat file1.dat >> file3.dat**
- **|** (“pipe”) redirects command output to another command; used to chain commands
  - **head -10 list.txt | tail -2**

# Exercise - redirect

For this exercise, we need to introduce the command `wc` which prints line (-l), word (-w), character (-m) or byte (-c) count of file

- In the `shell-lesson-data/exercise-data/proteins` directory
- Get a listing of all of the files
- Get the results of the use of the `wc` command of all of the files
- Get the results of the `wc` command of all the files giving only the number of lines
- Redirect the output of the last command to a file called `lengths.txt`
- View the contents of the file `lengths.txt`
- Use the `sort` command on the file `lengths.txt` and create a new file `sorted-lengths.txt` (What does the `sort` command do?)
- Use the `head` command to show the first 2 lines of both the `lengths.txt` and the `sorted-lengths.txt` files



# Exercise - append

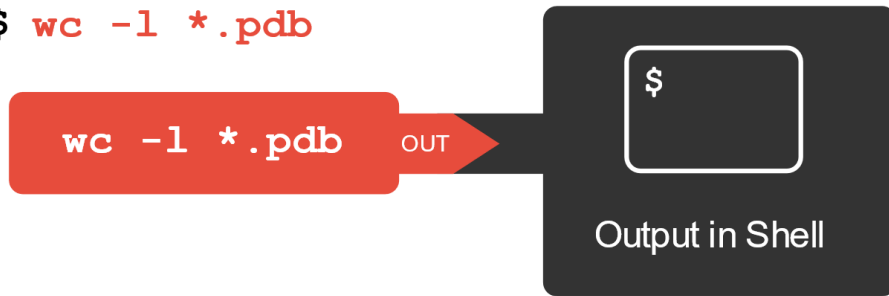
- Navigate to the `shell-lesson-data/exercise-data/animal-counts` directory and show the contents of the file `animals.csv`
- Using the `head` command create a file called `animals-subset.csv` that contains the first 3 lines of the file `animals.csv`
- Using the `tail` command create a file called `animals-subset.csv` that contains the last 3 lines of the file `animals.csv`
- Create a file `animals-subset2.csv` that contains both first 3 and the last 3 lines of the original `animals.csv` file

# Exercise - pipe

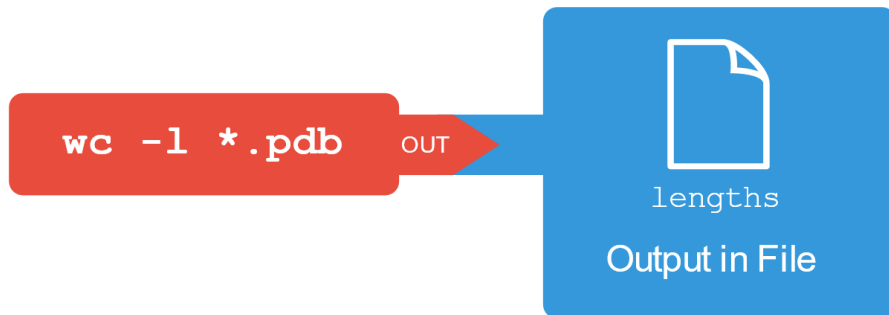
- In this exercise we are going to demonstrate the result of chaining commands
- In the `shell-lesson-data/proteins` directory we created a file `lengths.txt` and sorted this file.
- In this exercise we are going to go to this directory and do a set of commands, looking at the output
  - `wc -l *.pdb`
  - `wc -l *.pdb | sort -n`
  - `wc -l *.pdb | sort -n | head -n 1`
  - `wc -l *.pdb | sort -n | head -n 1 > shortest.txt`

# More about redirect and pipe

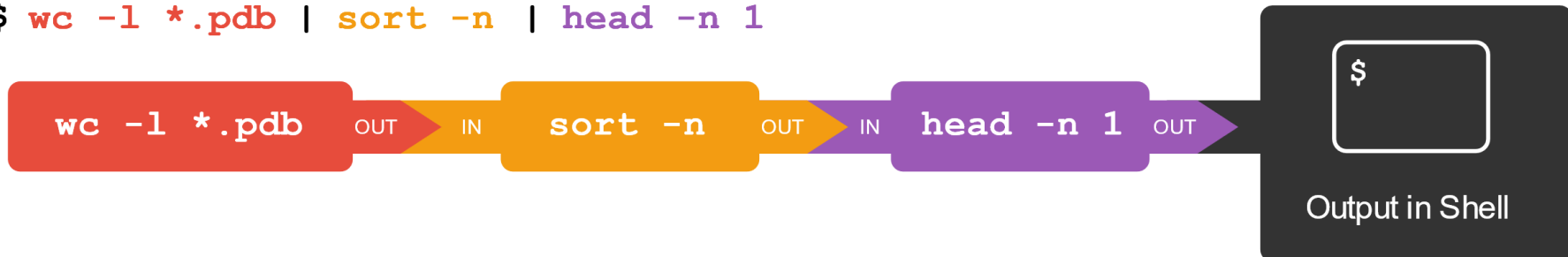
```
$ wc -l *.pdb
```



```
$ wc -l *.pdb > lengths
```



```
$ wc -l *.pdb | sort -n | head -n 1
```



# Nelle's Pipeline: Checking Files

- Nelle now has the first 17 results from the assay runs. As we mentioned, these are put in directory called **north-pacific-gyre**. These files are expected to be the same length and to be named in the consistent manner files **NENE01729A.txt**. The convention chosen are that there should be a A and B for each sample.
- How would you check that the data is what is expected?

# Have Questions?

- ❑ CHPC has an issue tracking system:  
helpdesk@chpc.utah.edu
- ❑ Some useful websites

<http://swcarpentry.github.io/shell-novice/>

<http://linuxcommand.org/>

<https://cvw.cac.cornell.edu/linux/default>