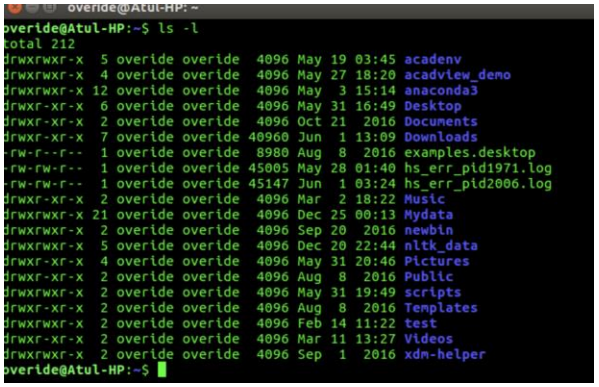
Introduction to Linux – Part 1

Brett Milash & Ashley Dederich

Research Consulting & Faculty Engagement

Center for High Performance Computing

helpdesk@chpc.utah.edu



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!)
- **Commands can take flags**/options that modify their behaviour

  - flags are formed with – (dash) and letter (sometimes --)
- **Commands and flags can also sometimes require an argument** – this defines the item upon which the command acts

Additional Shell Basics

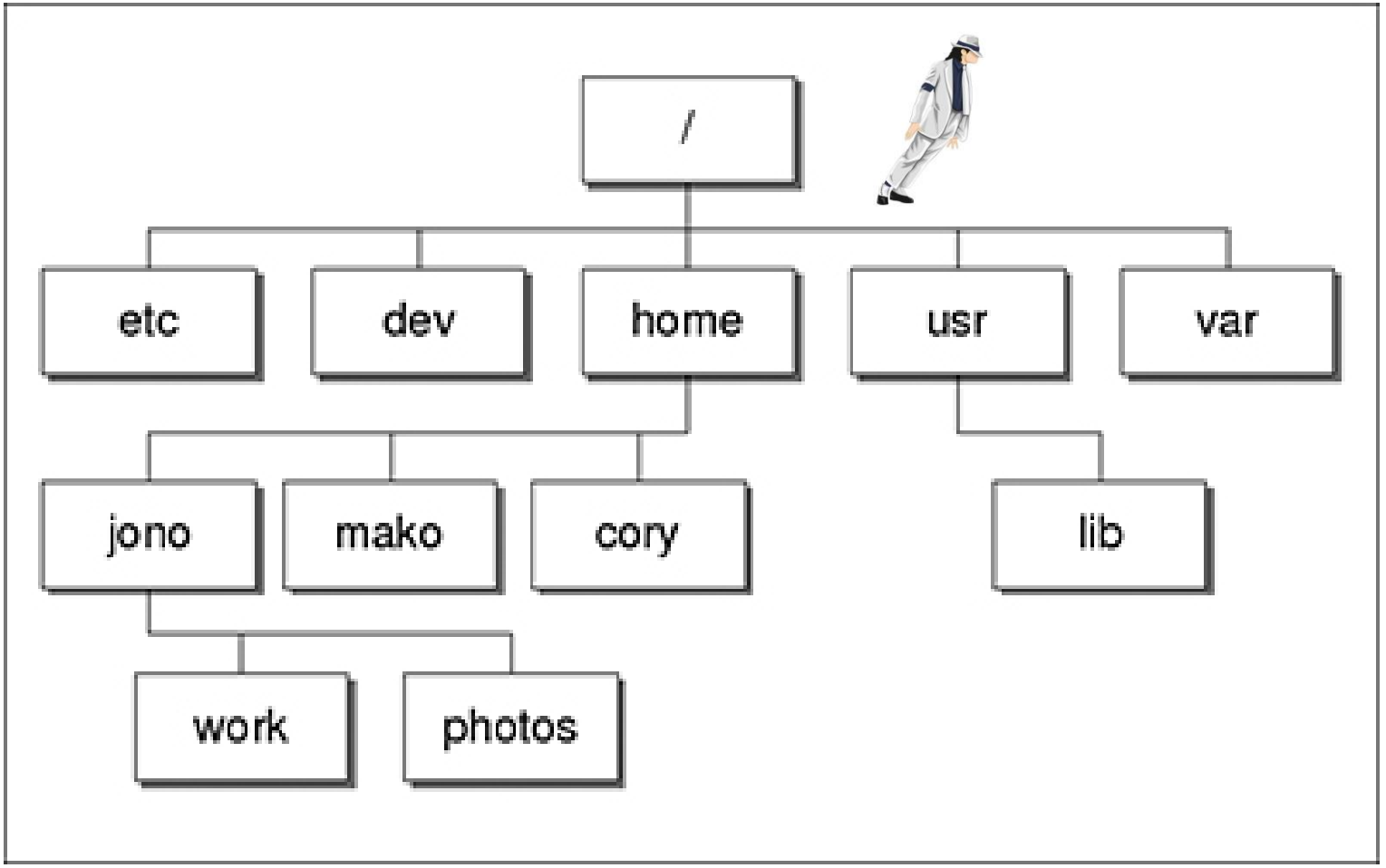
- Linux is case sensitive!
- CHPC offers two basic shells - slightly different command syntax
  - `csh/tcsh`
  - `sh/bash  (Bourne, Bourne again)`
- While many shell commands are the same between shell types – there are syntax and behaviour differences
- Your account comes with a script that is executed 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

Linux File System Directory Structure

- **The file system is a tree directory structure**

- levels are separated by **/** (forward slash, or slash)
  - Note – not the \ (backslash) used in Windows!
- **Topmost / --- 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
  - For 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

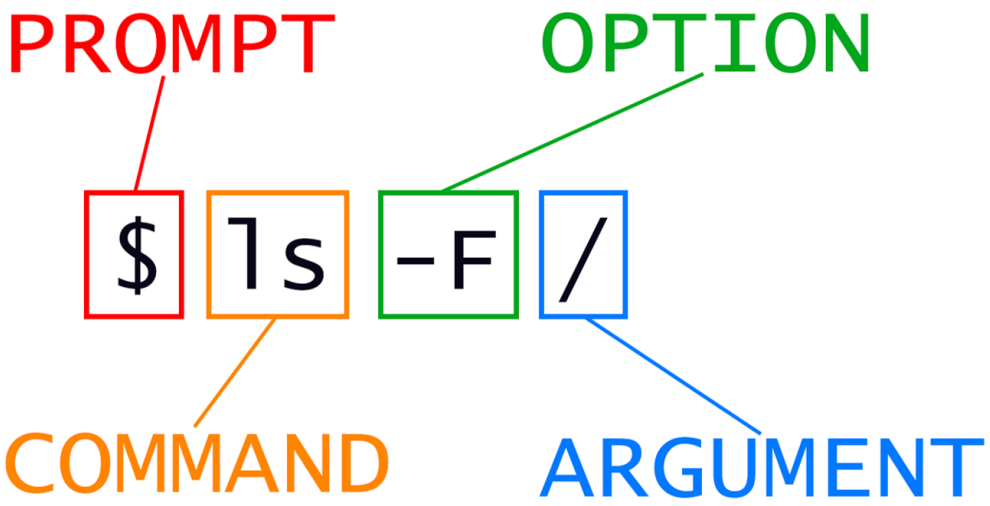


At CHPC cluster --- instead of **/home** we have **/uufs/chpc.utah.edu/common/home** under which we have all user directories

Login & Prompts

- When you first login you will see a prompt (the prompt is set by the login script)
  - [u1234567@notchpeak1:~]$
- 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

General Syntax of Shell Commands



Basic Directory Commands

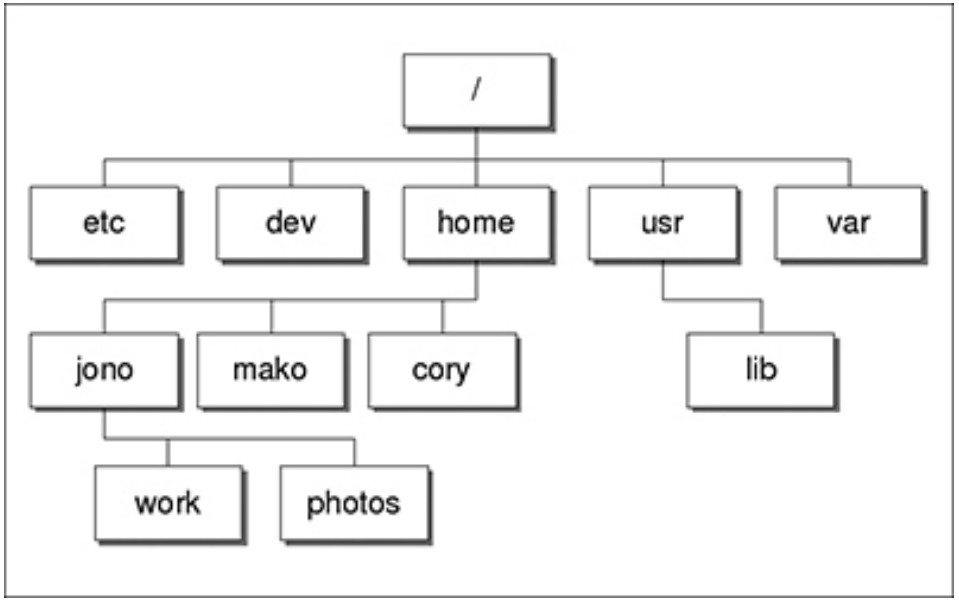
- **`ls`** – list contents of a directory
  - Flags to change output To see all flags
    - ls --help
    - man ls
- -l : long
- -a : All (including hidden files, also called dot files)
- -r : Reverse ordering while sorting
- -t : Timestamp
- -h : show memory usage in human-readable form

Linux File Permissions

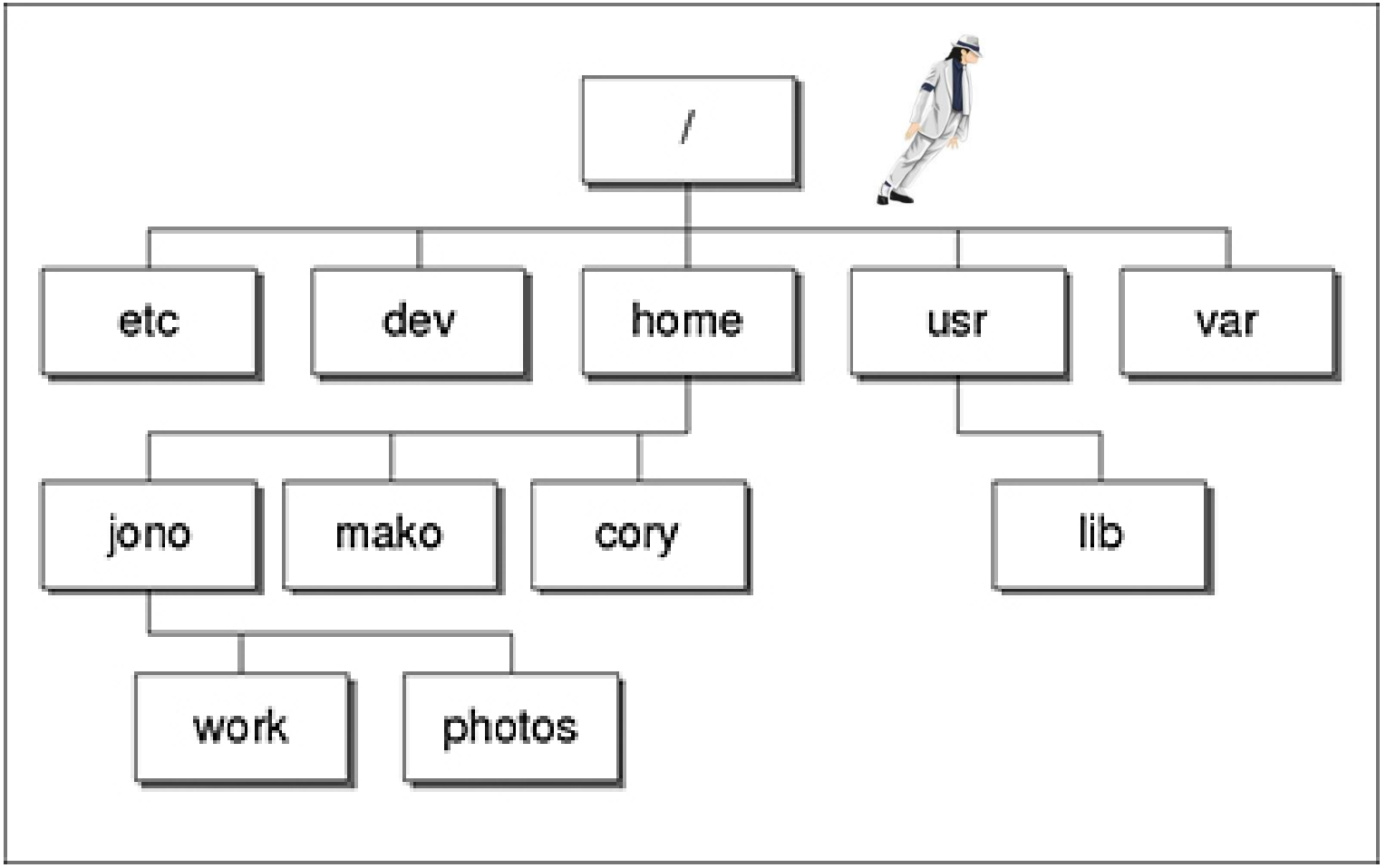
- Each Linux file belongs to a specific user (u) and a specific group (g)
- File permissions in format -rwxrwxrwx
- Shown with **`ls -l`**
  - -rw-rw-r-- 1 u0028729 chpc 86 Jul 30 02:41 notes.txt
  - -: file; d: directory
  - : user (u); : group (g); : world (o)
  - **r**: readable; **w**: writable; **x**: executable ("cd"-able for directory); -: no permission
    - examples: rwx; r-x; r--; ---;
- **`chmod`** – to change permissions of file or directory (**only User or Root/Admin can do it**)
- Examples:
  - **`chmod u=rwx file`** ← **Set User permissions to Read Write and Exec**
  - **`chmod g+x file`** ← **Grant Group Executable permission**
  - **`chmod o-rwx *.c.`** ← **Remove all permissions for world (not User, not Group)**
- Executable files (programs and scripts) must have executable permissions; directories must be executable in order to be able to cd into them
  - **`chmod +x *.R`**

Basic Directory Commands

- **`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



Linux Directory Structure



Files & Filenames

- Directories (folders) can contain files and other directories
- Filenames are often in the format of NAME.EXTENSION
  - Extensions are useful for telling you *and* the OS what type of file it is – IF you follow the conventions (txt, pdf, jpg, etc)
- Files that start with a “.” are hidden or “dot” files
- 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 -, =

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!

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)

Wildcards

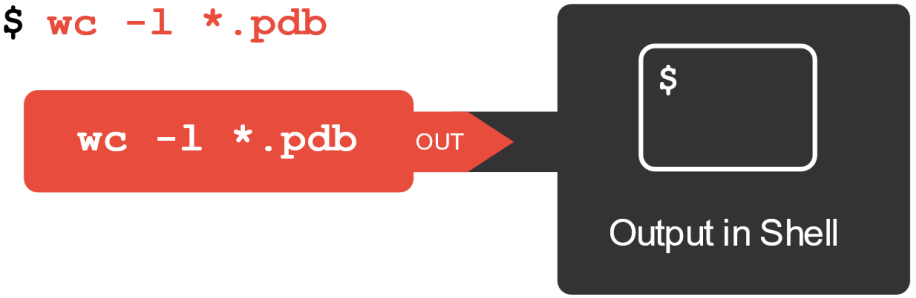
- multiple files can be specified via wildcards
- `*` - matches any number of letters including none
- `?` - matches any single character
- `[]` - encloses set of characters that can match the single given position
- `-` used within `[]` denotes range of characters

Command output redirection

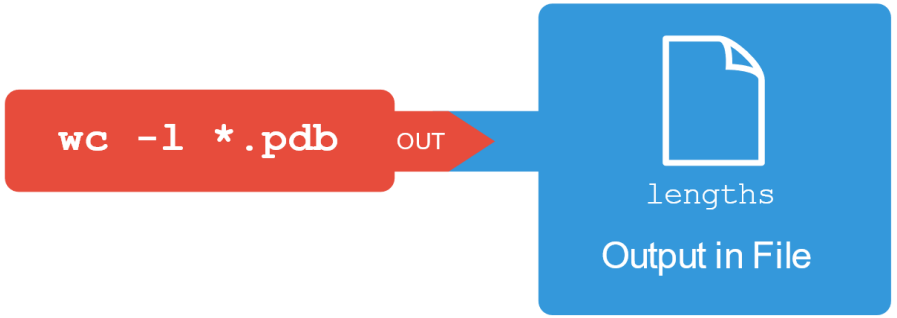
- **>** redirect output to a file (instead of to screen)
  - will create file if it does not exist
  - will overwrite the previous contents if it does exist
  - **`cat file1.dat > file4.dat`**
- **>>** append to a file
  - **`cat file1.dat >> file3.dat`**
- **|** (“pipe”) redirects command output to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

More about redirect and pipe

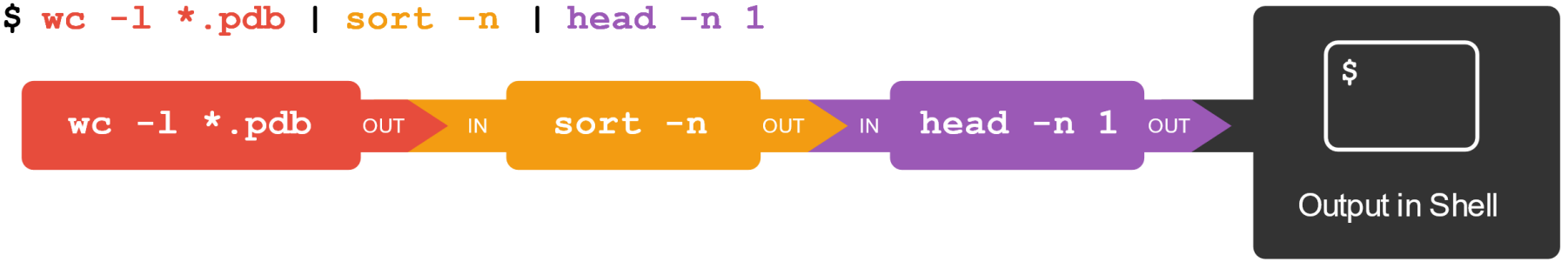
```
$ wc -l *.pdb
```




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




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



Loops

- Used when you want to preform the same action many times, such as on multiple files
- There are a number of ways you can do this
- One option
  - List multiple arguments for a command to act upon
    - **`head -n 3`** basilisk.dat minotaur.dat unicorn.dat
- Another option – do a loop with a for/do statement

Loop Terminology

- in bash syntax a loop looks like:

```bash
for thing in list_of_things
do
    operation_using $thing    # Indentation within the loop is not required, but aids legibility
done
```

- In this loop **thing** is a **variable**. During execution **$thing** is set to the first item in the list, the operation(s) is done, then it goes to the second and repeats the operation(s), etc until it reaches the last item in the list. Then the loop is exited.
- You can choose anything for **thing** (eg mything, item, myfile…) – however, your choice of the what to use for thing should help a person reading the file understand what the loop is doing and what it is acting upon
- Examples of **list_of_thing**: 2 4 6 8 10; ← a list of numbers

  {2..10..2}; {start..end..step} ← A number range

  fileA fileB fileC fileD; ← a list of filenames

  file*; ← a list of filenames represented by wildcard

  $variable or $(command) representing a list: $(ls) ← variable or command

Some other useful Linux commands

- **`cut`** – e.g. **`cut -d , -f 2,3 animals.csv`**
  - **`Example file: shell-lesson-data/exercise-data/animal-counts`**
  - **`-d (delimiter) -f (column ids)`**
  - Prints selected parts of lines from file to standard output (screen)
- **`du`** – e.g. **`du –h or du -sh`**
  - Scan a given file/directory (and subdirs) and report space usage; -s give summary of total usage, -h gives it in “human readable” format of K, M, G
- **`df`** – e.g. **`df –h`**
  - Overview of file system disk space usage (-h: human readable)
- **`ln`** – e.g. **`ln -s ~/bin/prog.exe prog1.exe`**
  - create a link between files (ln -s FILE LINK)

On your own – Use and explore options of these commands

Login Scripts & Environment Variables

- In your home directory are a number of dot files - `.bashrc` and `.custom.sh, .tcshrc` and `.custom.csh` Depending on your shell choice, the appropriate pair of these are executed during login.
- These set the environment (as environment variables) needed for you to work on CHPC resources
- Commands to check your environment: `env` or `printenv`
- Some important variables
  - $USER
  - $HOME
  - $PATH – paths to search for commands
  - $LD_LIBRARY_PATH – paths to search for libraries when linking a program (more on that later)

Processes

- A Process is a running Linux program
  - Each process has a PID (Process ID)
- **top** displays processes and resource usage in real time (Ctrl + C to quit)
  - top –u <user>
  - Ctrl + C to quit
- **`ps`** reports a snapshot of current processes
  - **`ps, ps x`** Display ALL of your processes
  - **`ps ax`** Display ALL processes
  - **`ps aux`** Display ALL processes (more detailed)
- **`kill PID`** kills the process with the specified PID
- **`killall processname`** kills all process with the processname
- **`kill -9 PID`** kills the process with the specified PID if a kill does not work

Monitoring processes/usage

- **uptime** – how long the system has been running
- **free** – free –h, memory and swap usage
- enhanced **top**
  - **atop** (available on CHPC clusters)
  - **htop** (available on CHPC clusters)
- **sar** – historical system usage report (cpu, memory, I/O…)

Moving Files To/From CHPC

- https://www.chpc.utah.edu/documentation/data_services.php
  - Can mount CHPC file systems on your local machine (Windows, Mac or Linux), must be on campus or using the campus VPN
  - Windows – there are graphical tools such as WinSCP
  - Mac, Windows, cloud options – cyberduck, another graphical tool
  - Linux
    - **scp** command (secure shell copy) – to copy files between linux systems
    - scp FILENAME UNID@CLUSTER.chpc.utah.edu:DIRECTORY/PATH
  - **wget**– to download from web with URL
    - **curl** is another option
  - For larger data sets – look into the Data Transfer Nodes (DTNs) and transfer tools such as **Globus**, see
    - https://www.globus.org/quickstart
    - https://www.chpc.utah.edu/documentation/data_services.php

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

Editors

There are many choices – a few are:

- nano

- vi/vim

- emacs

Nano Editor

To start either

nano

OR

nano filename

-- if filename exists, it will open file in editor; if it does not, this will be the name used when you save the file.

if you start nano without a filename it will prompt you for a name when you “WriteOut” using ^O (Ctrl + O)

^? → Ctrl + a specific Key; How to quit nano session: ^X → Ctrl + x

https://www.nano-editor.org/dist/latest/cheatsheet.html

Vim editor

- Start with the command **`vim`** or **`vim`** filename
- vi cheat sheet – linked on the presentation page https://www.chpc.utah.edu/presentations/IntroLinux3parts.php
- There is also a tutor program – start with command **`vimtutor`** which is a great tool to learn to use the program
- Insert Text: **i**
- Stop inserting text: **Esc**
- Down/Up: **j/k**
- Save File and Quit: **Esc + wq + Enter**
- Quit File and Discard Changes: **Esc + q! + Enter**

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