Introduction to Linux using the HPRC Portal

Xin Yang

High Performance
Research Computing
DIVISION OF RESEARCH

Course Outline

10:00 -10:15 Introduction and Accessing the system
10:15 -10:30 Hands-on Session 1

10:30-10:50 Bash shell, utilities, Directories and Files
10:50-11:05 Hand-on Session 2

11:05-11:15 Break

11:15-11:35 Wildcards, file attributes, Bash Environment Variables
11:35-11:50 Hands-on session 3

11:50-12:30 grep, file compression, redirecting io, ssh, file transfer, vi

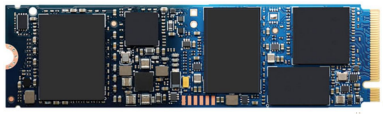
What is Linux?

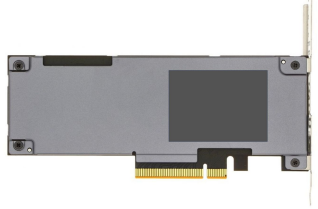
- 1st Unix OS (**1969** developed at the Bell laboratories)
  - Macintosh OS (1979)
  - DOS - Disk Operating System (1980, Tim Paterson)
  - Linux (**1991**, Linus Torvalds)
- Linux is a popular operating system
  - Stable, Fast, Secure and Powerful
  - Designed for **multi-user** and multi-tasking
  - Easy to share data and programs **securely**
- Command line is not user friendly
  - "Unix is user friendly, it is just particular about who its friends are."
- Available for almost all hardware.
- Common Linux Operating Systems
  - Ubuntu, Fedora Core, Centos, Red Hat, SUSE, etc

Shared Resources

- CPU (Central Processing Unit) - Allocation to a process based on a priority scheme
- Memory
  - RAM (Random Access Memory): Used for fast access to data of a program
  - SWAP: Slower because the program needs to read/write the data needed from the hard drive. Swapping refers to moving entire processes in and out of main memory to disk.
  - **free** is a linux command to show memory availability

- Disk (Hard Drive(s))
  - On small systems, the user normally has access to the entire disk space available in the home and scratch partitions.
  - On larger systems, the user is limited to the disk space allocated to users via a quota system.





Setting up an account

- Username/User ID - unique name on a machine - TAMU NetID on HPRC (*netid*)
- Password - Both State of Texas law and TAMU regulations prohibit the sharing and/or illegal use of computer passwords and accounts.
- Shell - a program that lets the user communicate with the Linux kernel.
  - Great information about shells: www.linfo.org/shell.html
  - **Bash shell (bash)** - most commonly used shell on Linux systems
  - Bourne shell (sh) – often used for system administration.
  - C shell (csh)
    - T-shell (tcsh) - historically, most commonly used shell on UNIX systems
  - Kourne shell (ksh) – most commonly used on IBM/AIX systems
  - See http://www.freebsd.org/ports/shells.html for a long list of shells (zsh, ash, dash, fish, mudsh, etc)

Directives used in this Lecture

Commands to type in will use the following:

- **Bold** words should be entered explicitly
- *Italicized* words are variable depending on the information that the utility needs
- commands for you to type in
- command output in

Accessing the system

- HPRC Portal:
  - https://portal.hprc.tamu.edu/
  - login with your HPRC account
- SSH (secure shell):
  - Encrypted communication
  - Windows:
    - https://hprc.tamu.edu/wiki/HPRC:MobaXterm
  - MacOS:
    - https://hprc.tamu.edu/wiki/HPRC:Access:MacOSX

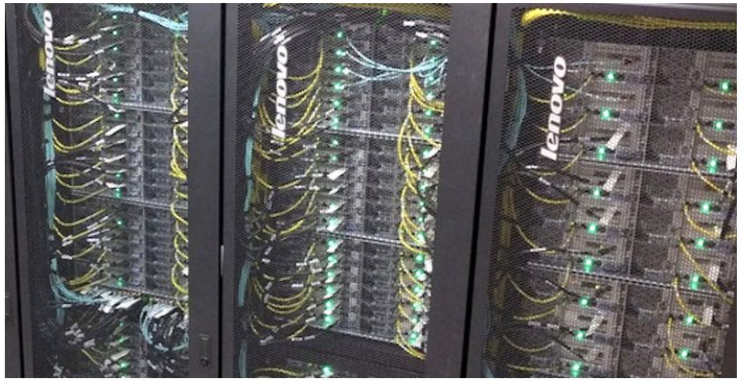
portal.hprc.tamu.edu

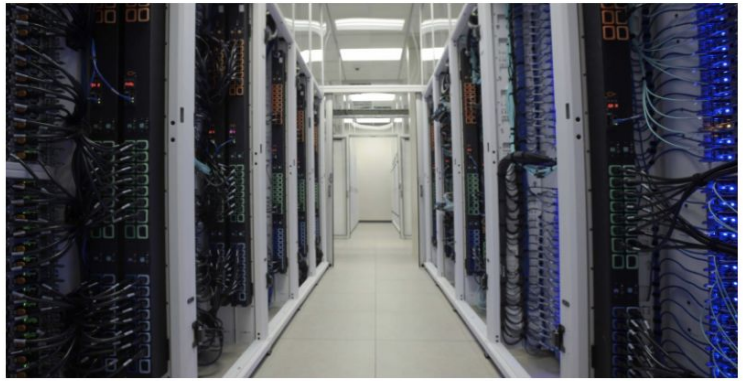
High Performance Research Computing

A Resource for Research and Discovery

TEXAS A&M UNIVERSITY

TAMU HPRC OnDemand Homepage





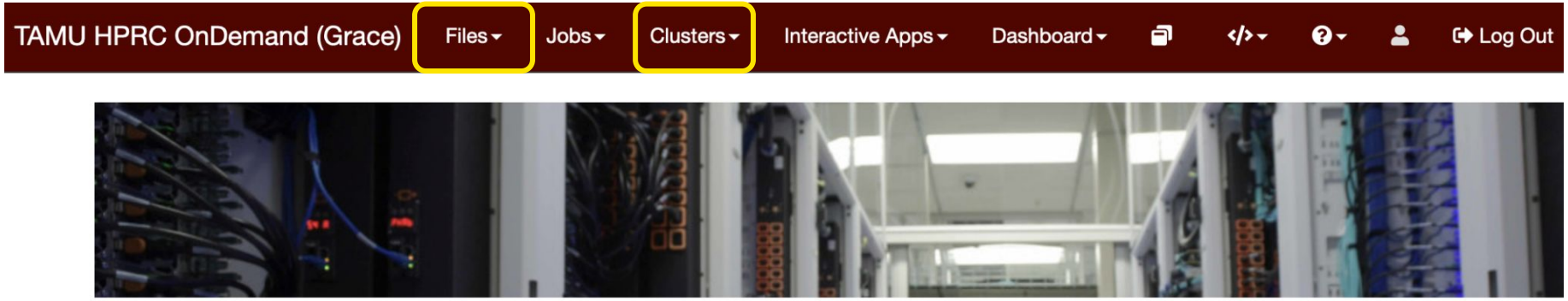
Terra OnDemand Portal

Grace OnDemand Portal

OnDemand Portal User Guide

Select “Grace OnDemand Portal”

Using the Portal



- **Files** > copy and edit files on the cluster's filesystems
- Jobs > submit and monitor cluster jobs
- **Clusters** > open a shell terminal (command line) on a login node
- Interactive Apps > start graphical software on a compute node
- Dashboard > view file quotas and computing account allocations

Hands-on Session 1

- logon to the portal
- navigate to the file menu
- turn on hidden files
- open a terminal

Bash Shell Control

- Prompting
  - Bash prompt can be defined by the PS1 variable:
  - **`PS1="[\u@\h \W] : "`**
  - [*username*@*hostname folder*] :
  - [*netid*@grace Linux] :
  - An active prompt means that the shell is ready for you to type a command.
- Command Interpretation and Execution
  - When a command is typed at the prompt, the Shell processes the command and sends it to the Linux kernel.
    - example: [*netid*@grace1 ~] : **`ls`**
    - [*netid*@grace ~] : is the prompt and **`ls`** is the command
    - **`ls`** is a command to list all the files in the current directory
    - more about commands later….
- Each shell has its own scripting language

A scripting language is a programming language that supports scripts: programs that automate the execution of tasks that could alternatively be executed command line. Scripting languages are often interpreted (rather than compiled).

Simple Utilities

`logout` or **`exit`** - closes a terminal or ssh session

`date` - displays the current date and time (not necessarily the correct date or time)

`clear` # clears your screen

`hostname` # prints the hostname to the screen

`whereis` *`command`* # find a program

`locate` *`command`* # find a file (program, dir, file, etc)

`ctrl-c` # (^c) interrupts a process abruptly

Directories and Files

Quick Q&A

• What is a directory?	Directory = folder
• Where am I?	pwd
• How do I move around?	cd
• Where could I go?	Anywhere you have permission

We'll talk about each of these and more in this section.
If you have any questions please feel free to stop and ask for clarification.

File and directory names

```
Commonly used:
A-Z
a-z
0-9
.
-   dash
_   underscore
```

- Do NOT use spaces in the file name
  - ("**`my data file.txt`**" vs "**`my_data_file.txt`**").
- File and directory names are case sensitive
- Avoid creating files on your Windows computer and copying to linux especially with spaces in the file name

```
Do NOT use:
   spaces or tabs
() parenthesis
" ' quotes
?  Question mark
$  Dollar sign
*  Asterisk
\  back slash
/  forward slash
:  colon
; semi-colon
 ampersand
@ & [ ] ! < >
```

Get a file from a URL

Use the **wget** command to get a file from a URL

```
wget https://hprc.tamu.edu/files/training/DOS_script.sh
```

```
--2022-03-03 21:43:09--  https://hprc.tamu.edu/files/training/DOS_script.sh
Resolving hprc.tamu.edu (hprc.tamu.edu)... 165.91.16.14
Connecting to hprc.tamu.edu (hprc.tamu.edu)|165.91.16.14|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 47 [text/x-sh]
Saving to: 'DOS_script.sh'

100%[======================================================================
==================================>] 47          --.-K/s   in 0s

2022-03-03 21:43:09 (4.82 MB/s) - 'DOS_script.sh' saved [47/47]
```

File Type - CRLF Line Terminators

Windows editors such as Notepad will add hidden Carriage Return Line Feed (CRLF) characters that will cause problems with many applications

```
cd
file DOS_script.sh
```

```
DOS_script.sh: ASCII English text, with CRLF line terminators
```

dos2unix command will convert the file to unix format

```
dos2unix DOS_script.sh
```

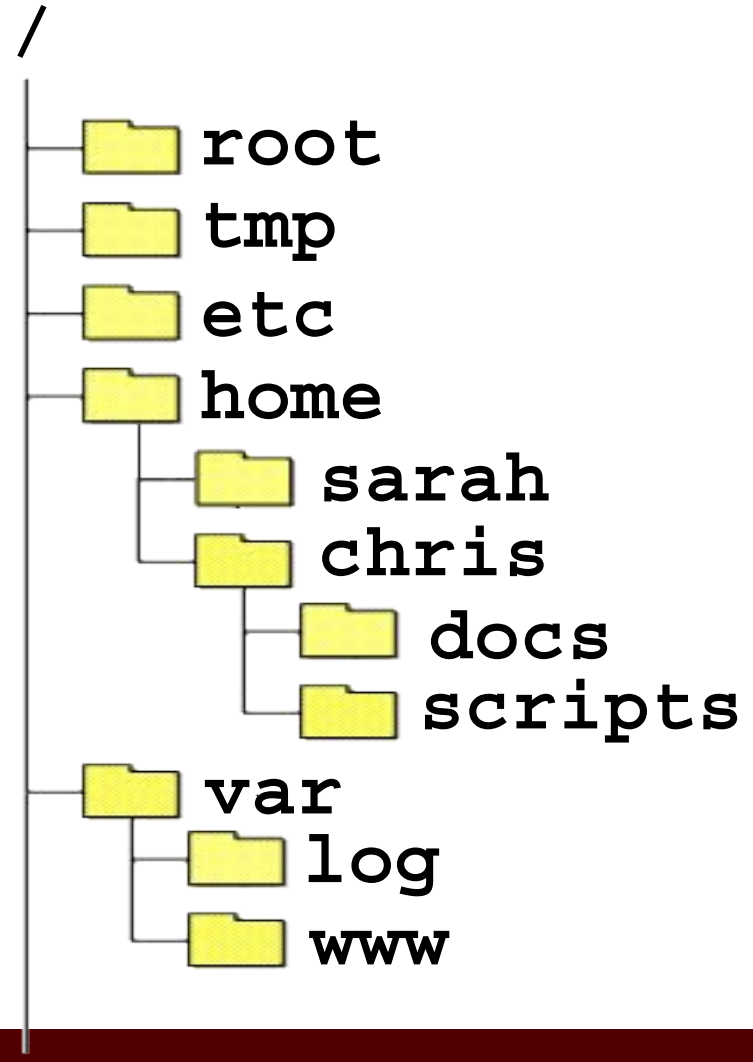
```
dos2unix: converting file DOS_script.sh to Unix format ...
```

```
file DOS_script.sh
```

```
DOS_script.sh: ASCII English text
```

File System Hierarchy

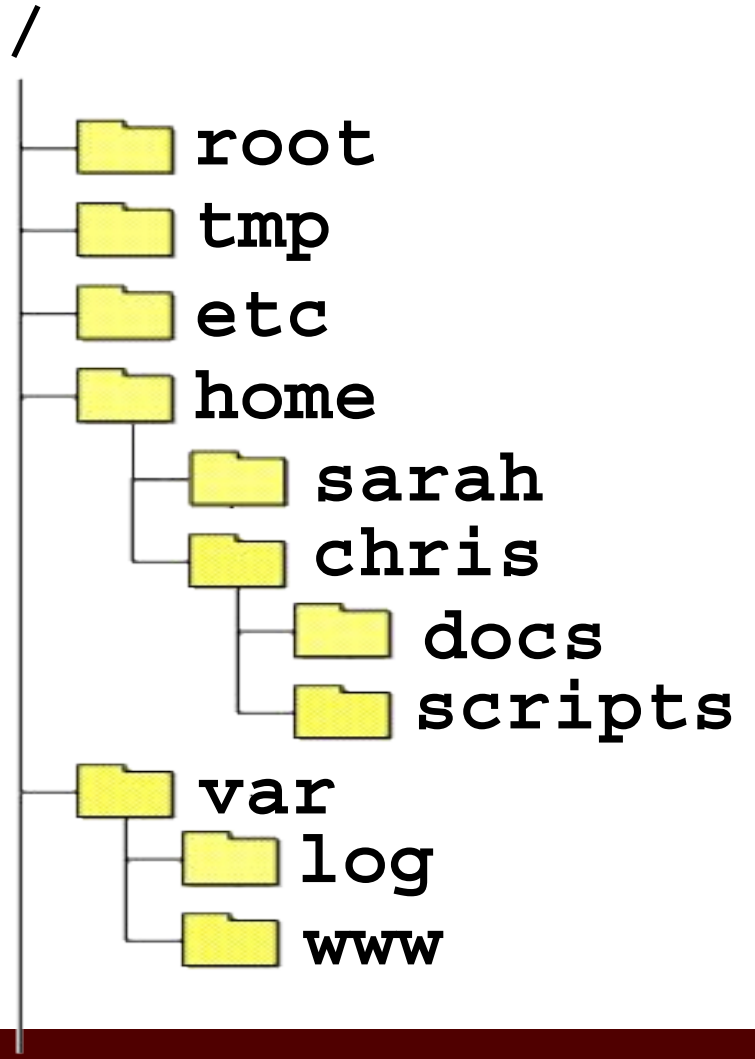
Finding your way around the Linux directory structure



/
/root
/tmp
/etc
/home
/home/sarah
/home/chris
/home/chris/docs
/home/chris/scripts
/var
/var/log
/var/www

File System Hierarchy

- **pwd** - **p**rints your current **w**orking **d**irectory
- **cd** - changes to your home directory (**c**hange **d**irectory)
- **cd** *name* - change directory to name
  - absolute pathnames (start with a forward slash **/**)
    - cd /home/chris/docs
  - relative pathnames (do NOT start with a /)
    - **.** current directory
    - **..** parent directory
    - **~** home directory
      - cd ../../tmp
      - cd ~
      - cd ~/docs
      - cd ~chris



Managing Files & Directories

Printing **directory** contents to the screen

- **`ls`** - lists contents of working directory
- **`ls`** *`dirname`* - lists the contents of the directory specified by dirname
- `ls -aCFl`
  - flags
    - -a print hidden files
    - -l print long listing
    - -F print a special character after special files
    - to find all possible flags, use the command: `man ls`
- **`tree`** - recursive directory listing

Managing Files & Directories

Printing ASCII (text) file contents to the screen

- **`less`** *`filename`*
- **`more`** *`filename`*
- **`page`** *`filename`*
- **`cat`** *`filename`*
- **`cat -A`** `filename`
  - `show hidden characters`
- **`head -`***`n filename`*
  - n is an integer
  - displays the first *n* lines
- **`tail -`***`n filename`*
  - displays the last *n* lines
- **`tail -f`** *`filename`*
  - Display the last 10 lines of a file and waits for new lines, ctrl-c (^c) to exit.

Managing Files & Directories

- Making a directory (dir)
  - **`mkdir`** *`dirname`* (creates a directory in the current dir)
  - `mkdir tmp` (creates the directory tmp in the current dir)
  - `mkdir ~/tmp` (creates the directory tmp in your home dir)
  - `mkdir /home/`*`netid`*`/tmp` (created the directory tmp in /home/*netid*)
- Rename a directory
  - **`mv`** *`olddirname newdirname`*

Managing Files & Directories/Folders

- Renaming a file
  - **mv** *oldfilename newfilename* (note: new **cannot** be a directory name) You need to specify the location of *oldfilename* and *newfilename*. This command specifies the *oldfilename* and *newfilename* are in the current directory because there is nothing in front of the names.
- Move a file into a new directory
  - **mv** *filename dirname* (note: *dirname* must be a directory that already exists.)
  - retains the filename but moves it to the directory *dirname*
  - You can rename the file while moving it to a new directory:

    mv *oldfilename dirname/newfilename*
- Safe mv
  - **mv -i** *oldfilename newfilename*
  - -i is a flag that modifies the way mv behaves. In this case –i tells the command to prompt you for permission if you are about to overwrite a file.

Managing Files & Directories

- Making a copy of a file
  - **`cp`** *`oldfilename newfilename`*
    - Makes a copy of the file named *oldfilename* and names it *newfilename* in the current directory
    - Note: *newfilename* cannot be the name of a directory
- Copying a file to a new directory
  - **`cp`** *`filename dirname`*
    - Makes a copy of the file named *filename* to the directory named *dirname*
    - Note: *dirname* must already exist
- Safe copy
  - **`cp -i`** *`oldfilename newfilename`*
    - will prompt you if you are about to overwrite a file named *newfilename*

Managing Files & Directories

- Copying a directory
  - **`cp -R`** *`olddirname newdirname`*
    - Makes a complete copy of the directory named *olddirname* including all of its conents, and names it *newdirname* in the current directory
    - Note: *newdirname* cannot be the name of a directory that already exists

Managing Files & Directories

- Deleting a file
  - **`rm`** *`filename`*
    - Deletes the file named *filename*
- Safe delete
  - **`rm -i`** *`filename`*
    - will prompt you for confirmation before deleting *filename*
- Deleting a directory
  - **`rmdir`** *`dirname`*
    - Deletes an empty directory named *dirname*
  - **`rm -r`** *`dirname`*
    - removes the directory named *dirname* and all of its contents.
- **Warning! Once a file is deleted or overwritten it is gone.** Be VERY careful when using wildcards. `rm -r *` will remove everything from that directory and down the hierarchy!

Hands-on Session 2

make two directories:

```
mkdir temp1
```

```
mkdir temp2
```

move the DOS_script.sh file to temp1:

```
mv DOS_script.sh temp1
```

show the directory hierarchy using the tree command:

```
tree
```

Wildcards (globbing)

*****	* matches any character(s)
?	? matches one character
[…]	matches a single character for a specified range of characters within the brackets
{...,...}	matches a list of patterns separated by a comma within the curly brackets

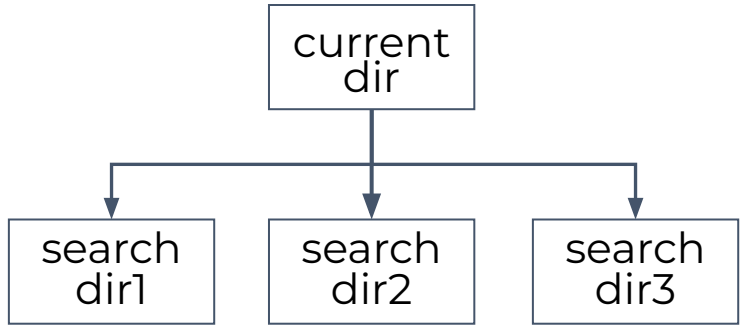
Examples

- **mv proj1* ~/Project1**
  - moves all files beginning with proj1 into dir Project1
  - the dir Project1 must already exists in your home dir
- **ls proj?.log**
  - lists all files where ? can be any one character
- **mv enzyme[12].com enzyme**
  - moves enzyme1.com and enzyme2.com into dir enzyme
- **mv project{*.com,*.log,*.txt} project1-5**
  - moves all files that start with project and end with .com, .log, or .txt to the directory project1-5 that already exists.

Managing Files & Directories

Searching for a file or directory

- **whereis** *filename*
- **locate** *filename*
- **find . -name** '*search string*'
  - `find . -name '*test1*'`
  - searches for any file or directory with the string test1 in it from the current directory and down the hierarchy (**-iname** makes the search case insensitive)



File Attributes

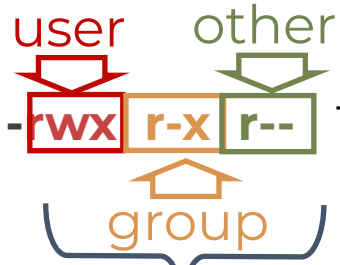
`ls -l` lists the files in the dir in long format

Note: the flag is the **`letter l`** and not the number `1`

-rwxr-xr-- 1 training lms 30 Oct 28 13:16 Molden

1	hard link count
training	file owner
lms	group ID
30	file size
Oct 28 13:16	time the file was last modified
Molden	filename

File Attributes



1 training lms 30 Oct 28 13:16 Molden

User – read, write & execute
Group – read & execute
Other – read only

groups of 3 for user, **group**, & others
r permission to read
w permission to write
x permission to execute
- permission is denied

leading character
- text
d directory
l link

Example:
-rwx**r-x**r-- 1 training **lms** 30 Oct 28 13:16 Molden
User has read, write and executable permission
Group has read and executable permission but not write permission
Other has read permission but not write or executable permission

Permissions

- To change the read, write and executable permission for users (u), group (g), others (o) and all (a)
- **`chmod u+x`** *`filename`* (or *dirname*)
  - adds executable permission for the user
- **`chmod og-r`** *`filename`* (or *dirname*)
  - remove read permission for group and others
- **`chmod -R a+rx`** *`dirname`*
  - give everyone read and executable permission
- from *dirname* and down the hierarchy
- `chmod u=rwx` *`filename`*
  - sets the permission to rwx for the user
- `chmod g=` *`filename`*
  - sets the permission to --- for the group
- You can also use numbers
- r = 4, w = 2, and x = 1, - = 0
  - chmod 755 *filename* (result -rwxr-xr-x)
  - chmod 600 *filename* (result -rw-------)

---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

Ownership/Groups

- To change the group
  - **chgrp** *groupname filename* (or *dirname*)
    - Changes the group for *filename* or for *dirname* but not for the files contained within *dirname*
  - **chgrp -R** *groupname dirname*
    - Changes the group for all of the files and directories down the hierarchy from *dirname*
    - Example: **chgrp -R lms training**
- Change owner
  - **chown** *username filename* (or *dirname*)
    - Changes the owner of *filename* or *dirname* but not for the files contained within *dirname*
  - **chown -R** *username dirname*
- The chown and chgrp command is not allowed for users by default on many Linux OS's
- Change owner and group
  - **chown** *username:groupname filename*
  - **chown** *username.groupname filename*

Bash Environment Variables

- Environment variables store information that is used across different processes in a Linux system.

- Use all caps for Bash Environment variable. **A-Z 0-9 _**
- Use lowercase for the variables that you create. **a-z 0-9 _**

  - **HOME** Pathname of current user's home directory
  - **PATH** The search path for commands.

- Use the **echo** command to see the contents of a variable

```
echo $HOME
```

```
/home/netid
```

The Search PATH

- The shell uses the **PATH** environment variable to locate commands typed at the command line
- The value of PATH is a colon separated list of full directory names.
- The PATH is searched from left to right. If the command is not found in any of the listed directories, the shell returns an error message
- If multiple commands with the same name exist in more than one location, the first instance found according to the PATH variable will be executed.

```
echo $PATH
```

```
/usr/lib64/qt-3.3/bin:/sw/local/bin:/usr/local/bin:/usr/bin:/usr/loca
l/sbin:/usr/sbin:/usr/lpp/mmfs/bin:/home/netid/.local/bin
```

- Add a directory to the PATH for the current Linux session

```
export PATH=$PATH:/home/netid/bin
```

Customizing the Environment

Two important files for customizing your Bash Shell environment

`.bashrc` (pronounced dot bashrc)

contains aliases, shell variables, paths, etc.

executed (sourced) upon starting a non-login shell.

`.bash_profile` (dot bash_profile)

also can contain aliases, shell variables, paths, etc

normally used for terminal settings

executed (sourced) upon login

if **.bash_profile** doesn't exist, the system looks for **.profile** (dot profile)

`. .bashrc` (or **`source .bashrc`**)

Executes the commands in the .bashrc file

The ␣ character will be used to represent a space

.bash_profile file contents

```
# Get the aliases and functions
    if [ -f  ~/.bashrc ]; then
            .  ~/.bashrc
    fi

# User specific environment and startup programs
    PATH=$PATH:$HOME/.local/bin:$HOME/bin
    export PATH

# personal aliases
    alias h= "history|more"
    alias m="more"
    alias ll= "ls  -la"
    alias ls= "ls -CF"
    alias rm= "rm  -i"
    alias cp= "cp  -i"
    alias mv= "mv  -i"
    alias x="chmod u+x"

# A line that begins with a # is a comment
```

.bash_profile file contents

```
# Syntax to set a local variable
#     varname=value
# Syntax to set a global variable
#     export varname=value
# Syntax to set an alias
#    alias name="value"
# Syntax to create a function
#     function name() { command ; }


function cc() { awk -f cc.awk "$@".log>"$@".cc ; }
# If you type cc test at the prompt, the following command will be executed:
# awk -f cc.awk test.log > test.cc
```

Editing an text (ASCII) file

There are many editors available under Linux.

- File editor in the HPRC Portal
- Text mode
  - nano (simple)
  - vi or vim (more advanced)
  - emacs (more advanced)
- Graphic mode (require remote graphics support - X11)
  - gedit
  - xemacs
  - gvim
- Be aware that a text file edited under Windows editors will most likely add CRLF characters. Use **`dos2unix`** to convert a DOS/Windows edited text file to Unix format.

Hands-on Session 3

- Create a new alias in your .bash_profile file named **simple** that executes the command: echo I succeeded in created a simple alias
- active your new alias **`. .bash_profile`**
- Use your new alias. Type: **`simple`**
- Make a directory named **`myapps`** in your home directory
- Create a file (your choice of a name) in `myapps` with the following content:
  - echo I succeeded in adding `myapps` to my path
- Change the permissions of `myapps/`*`filename`* to allow execution (replace *filename* with the name that you used):
  - **`chmod u+x $HOME/myapps/`***`filename`*
- Run *filename* by typing *filename* (you should get an error message)
- add **myapps** directory to your PATH by editing your `.bash_profile` file
- active your new `PATH` **`. .bash_profile`**
- Run filename by typing: *filename*

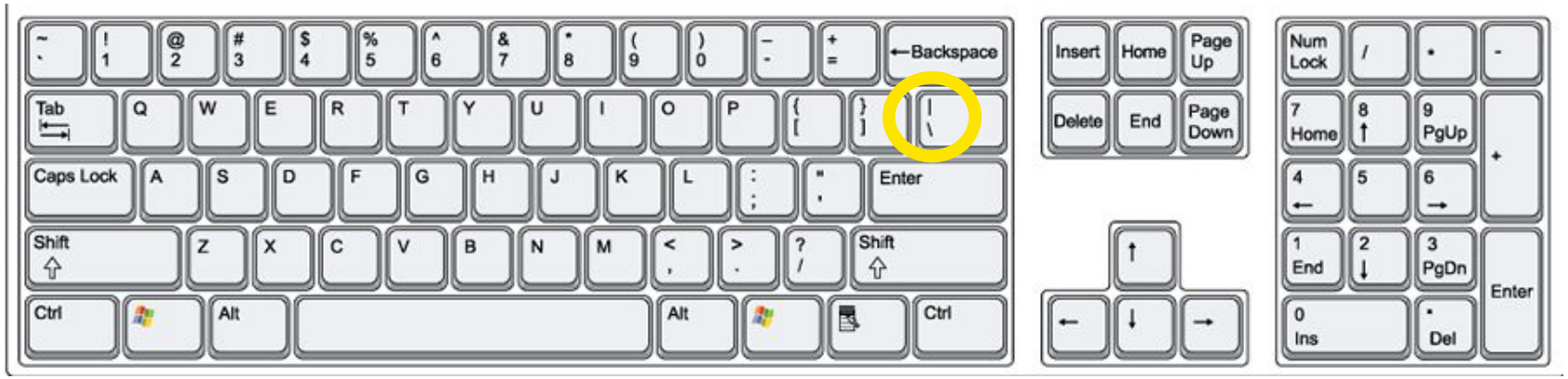
Searching File Contents

- **`grep`** *`search-pattern filename`* - searches the file *filename* for the pattern *search-pattern* and shows the results on the screen (prints the results to standard out).
  - `grep Energy run1.out`
    - searches the file run1.out for the word Energy
    - grep is **case sensitive** unless you use the **-i** flag
  - `grep Energy *.out`
    - searches all files in that end in .out
  - `grep "Total Energy" */*.out`
    - You must use **quotes** when you have blank spaces. This example searches for Total Energy in every file that ends in .out in each directory of the current directory
  - **`grep -R`** `"Total Energy" Project1`
    - Searches **recursively** all files under Project1 for the pattern Total Energy

Searching File Contents

egrep '*pattern1*|*pattern2*|*etc*' **f***ilename*

- searches the file filename for **all patterns** (*pattern1*, *pattern2*, etc) and prints the results to the screen.
- The **|** character is called a **pipe** and is normally located above the return key on the keyboard.
- `egrep 'Energy|Enthalpy' *.out`
  - searches for the word Energy or Enthalpy in every file that ends in .out in the current directory.



Compressing Files

- Compressing files
  - **`gzip`** *`filename`*
    - zips-up filename and creates filename.gz
  - **`gzip -v`** *`filename`*
    - zips-up filename in a verbose manner (tells you % compression)
  - **`gzip -r`** *`dirname`*
    - zips-up all files down the hierarchy from dirname
  - **`gunzip`** *`filename`***`.gz`**
    - unzips filename.gz and creates filename
  - **`bzip2`** *`filename`*
    - zips-up (compresses) filename and creates filename.bz2 (or .bz or .bzip2)
  - **`bunzip2`** *`filename`***`.bz2`**
    - unzips filename

Archiving Files/Directories

- **`tar -xpvf`** *`filename`***`.tar`**
  - Extracts the contents of filename.tar
- **`tar -cpvf`** *`filename`***`.tar`** *`filenames`* (or *dirnames*)
  - Archives filenames and/or dirnames into the file filename.tar
  - It is best to zip-up your files before archiving them.
- some of the tar flags
  - -c creates a new archive
  - -x extract files and/or directories form the archive
  - -p preserve protection information
  - -v verbose
  - -f working with files
  - -t lists the table of contents for an archive

ZIP command

- **`zip`** *`filename`***`.zip`** *`filenames`*
  - Zips and archives filenames into the file *filename***.zip**
- **`zip -r`** *`filename`***`.zip`** *`dirname`*
  - Zips and archives files in *dirname* and down the heirarchy into the file *filename***.zip**
- **`unzip`** *`filename`***`.zip`**
  - Extracts the contents of *filename***.zip**
- some of the tar flags
  - -v verbose
  - -l lists the table of contents for a zip file
  - -m delete the original files

Redirecting Input and Output

- **>** Redirects output
  - *command*`>`*outputfilename*
  - **`ls -al>list-of-files.txt`**
  - `>>` symbol appends to the end of the file instead of overwriting it.
  - **`ls -al>>list-of-files.txt`**
- **<** Redirects input
  - *program*`<`*inputfile*
  - `g16<run1.com`
  - output would go to standard out (stdout)
- Redirecting input and output together and running in the background
  - *program*`<`*inputfilename*`>`*outputfilename*`&`
  - `g16<run1.com>run1.log&`

Pipes

- Pipes **|**
  - takes the output of one command and sends it to another
  - **`ls|more`**
  - **`ls|less`**
    - List the files one page at a time
  - **`grep Energy run1.out|grep HF`**
  - **`grep Energy run1.out|grep HF>HF_output.txt`**
    - Searches a file named run1.out for the word Energy and then searches for the word HF in the lines that have the word Energy. The resulting information is then sent to a file named HF_output.txt

history, !, ↑, ↓, & tab completion

- **`history`**
  - The history command will list your last n commands (n = integer).
- **`!!`** # repeats your last command
- **`!`***`n`* # repeats the nth command
  - You can find the number of the command using history
- **`!`***`name`* # repeats the last command that started with name
- You can use the up (↑) and down (↓) arrow keys to scroll through previous commands
- `Tab` # will try to complete the rest of the file/directory name you are typing
  - If you have three files that start with x (xrun15 xrun16 and xrun17) then typing x and then tab will result in xrun1 at the prompt and you would have to type in the last character. On some systems, if you hit the tab key twice it will result in xrun1 at the prompt and list the 3 files that match.

Managing Disk Usage

- Most HPC systems impose a quota system for users
  - displays your disk allotment and usage: **showquota**
- `df -h` #displays the available file systems in the easiest readable unit.

```
[netid@grace4 ~]$ df -h
```

```
Filesystem                                                Size  Used Avail Use% Mounted on
devtmpfs                                                  189G     0  189G   0% /dev
tmpfs                                                     189G  1.2M  189G   1% /dev/shm
tmpfs                                                     189G  181M  188G   1% /run
tmpfs                                                     189G     0  189G   0% /sys/fs/cgroup
/dev/mapper/os_vg-root                                     80G   13G   68G  16% /
/dev/sda2                                                1014M  382M  633M  38% /boot
/dev/sda1                                                  50M   12M   39M  23% /boot/efi
/dev/mapper/os_vg-var                                      20G  1.6G   19G   8% /var
/dev/mapper/nvme1_vg-tmp                                  550G   16G  534G   3% /tmp
10.73.170.1@o2ib:10.73.170.2@o2ib:/fs01/home               10G  288K   10G   1% /home
10.73.170.1@o2ib:10.73.170.2@o2ib:/fs01/sw                4.5P  922T  3.5P  21% /sw
10.73.170.1@o2ib:10.73.170.2@o2ib:/fs01/scratch           4.5P  922T  3.5P  21% /scratch
```

- **du_-sh** prints your disk usage from the current directory and down (may take some time to complete)

Managing Processes

- **`top`** shows all processes in a table
  - **`q`** will exit the top process

```
top
```

```
top - 21:23:11 up 27 days, 11:16, 32 users,  load average: 2.05, 1.24, 1.08
Tasks: 1344 total,   2 running, 1327 sleeping,  15 stopped,   0 zombie
%Cpu(s):  7.4 us,  1.5 sy,  0.0 ni, 83.5 id,  7.7 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 39448889+total, 15868598+free, 62225532 used, 17357736+buff/cache
KiB Swap: 16777212 total,  8135384 free,  8641828 used. 33076313+avail Mem

   PID USER      PR  NI  VIRTRES SHR S  %CPU %MEM    TIME+ COMMAND
155988 netid     20   0  801400  74932  16192 S  42.8  0.0   0:02.00 orca_gtoint_mpi
155991 netid     20   0  794264  73920  16000 S  42.8  0.0   0:01.96 orca_gtoint_mpi
155985 netid     20   0  777460  73428  15376 S  37.5  0.0   0:01.83 orca_gtoint_mpi
155986 netid     20   0  794648  73968  15632 S  37.5  0.0   0:01.85 orca_gtoint_mpi
155939 netid     20   0  155316   3716   1628 R   1.3  0.0   0:00.22 top
141985 netid     20   0  163224   2664   1308 S   0.0  0.0   0:00.08 sshd
141993 netid     20   0  117552   4204   1804 S   0.0  0.0   0:00.52 bash
155922 netid     20   0  436064  58880  10952 S   0.0  0.0   0:00.06 orca
155923 netid     20   0  196124  16732  12408 S   0.0  0.0   0:00.29 mpirun
```

Managing Processes

- **`ps -u`** *`netid`* (list all of the processes for username)

```
ps -u netid
```

```
  PID TTY          TIME CMD
26780 ?        00:00:00 sshd
26781 pts/6    00:00:00 bash
27756 pts/6    00:00:00 gedit
28362 pts/6    00:00:00 ps
32432 ?        00:00:00 sshd
32433 pts/3    00:00:00 bash
32626 pts/3    00:00:00 vim
```

- **`kill`** *`pid`* kills the process with pid (process id number (PID)) nicely
- **`kill -9`** *`pid`* kills the process with pid without remorse – not nice or clean…
- To kill the gedit process: `kill 27756`
- Check to see if it is gone (`ps –u` *`netid`*) and if it is not, use: `kill -9 27756`

Computer Networking

- Secure Shell (ssh) - Access a remote machine through a secure encrypted protocol
  - **`ssh`** *`netid@remotehostname`* (username is different on the remote machine)
  - **`ssh`** *`remotehostname`* (username is the same on the local and remote machines)
    - `ssh grace.hprc.tamu.edu`
    - `ssh grace`
    - `ssh netid@grace.hprc.tamu.edu`
    - `ssh netid@grace`
    - The first time that you ssh to a machine from the local host, it will ask you for permission. You must type yes to continue (y will not work).
    - You will be prompted for your password
  - For remote graphics, you will need to ssh with the -X or -Y flag
    - **`ssh -X`** *`netid@remotehostname`*

Computer Networking

```
[netid@grace1 ~]$ ssh grace2.hprc.tamu.edu
```

```
Warning: Permanently added 'grace2.hprc.tamu.edu,128.194.35.40' (ECDSA) to the list of
known hosts.
**************************************************************************
This computer system and the data herein are available only for authorized purposes by
authorized users. Use for any other purpose is prohibited and mayresult in disciplinary
actions or criminal prosecution against the user. Usagemay be subject to security testing
and monitoring. There is no expectation of privacy on this system except as otherwise
provided by applicable privacy laws.Refer to University SAP 29.01.03.M0.02 Acceptable Use
for more information.
**************************************************************************
Password:
Duo two-factor login for netid
Enter a passcode or select one of the following options:
1. Duo Push to XXX-XXX-1564
2. Phone call to XXX-XXX-1564
3. SMS passcodes to XXX-XXX-1564
Passcode or option (1-3): 1
Success. Logging you in…

netid@grace2 ~]$
```

File Transfer Options

- Command line options: **`scp`, `sftp`, `rclone`, `rsync`**
- Use HPRC's fast transfer nodes for high-speed transfers
- For **small files** of less than 2GB
  - Use HPRC's Portal
  - MobaXterm from your computer
  - HPRC Galaxy for bio-researchers
- For **large files** sized hundreds of GB to greater than 1TB
  - Globus Connect website
  - ftp transfer for HPRC Galaxy

For more details and options please visit
https://hprc.tamu.edu/wiki/HPRC:File_Transfers

Secure File Transfer Protocol (sftp)

- **`sftp`** is used to transfer files between unix/linux machines
- **`sftp`** *remotehostname* or **`sftp`** *username@remotehostname*
  - sftp will ask you for your password and the first time you sftp to a machine it will ask you for permission. You must type **yes** to continue (y will not work).
- commands used in the sftp session
  - **`get`** *filename* - copies *filename* from the remote machine to the local machine.
    - Wildcard usage: get *.out get all of the files that end in .out automatically.
  - **`put`** *filename* - copies *filename* from the local machine to the remote machine.
    - Wildcard usage: mput *.out put all of the files that end in .out automatically.
  - **`ls`** - list the contents of the remote machine directory
  - **`lls`** - list the contents of the local machine directory

Secure File Transfer Protocol (sftp)

- commands used in the sftp session (continued)
  - **cd** *dirname* - changes the remote machine directory
  - **lcd** *localdir* - change the local machine directory
  - **mkdir** *dirname* - makes a dir *dirname* on the remote machine
  - **lmkdir** *dirname* - makes a dir *dirname* on the local machine
  - **pwd** - prints the working directory of the remote machine
  - **lpwd** - prints the working directory of the local machine
  - **bye** or **quit** - exits an sftp session.
  - **!***command* - executes a local shell command (i.e. hostname)

```
[netid@grace1 ~]$ sftp terra.tamu.edu
    sftp> pwd      Local working directory: /home/netid
    sftp> lpwd     Remote working directory: /general/home/netid
    sftp> bye
```

Secure copy (scp)

- **scp** *filename username@remotehostname:remotepath*
  - `scp run1.out netid@grace.hprc.tamu.edu:`
    - Makes a copy of run1.out located on the local machine to your home directory on grace
  - `scp run1.out grace.hprc.tamu.edu:/scratch/user/netid/`
    - Makes a copy of run1.out to /scratch/user/*netid* instead of the home directory. This syntax assumes that your username is the same on both machines
- **scp** *username@remotehostname:filename localpath*
  - Copies a file from the home directory on the remote host to the current directory on the local machine
- Useful flags:
  - `-r` recursively copy an entire directory (not suggested)
    - **scp -r** *dirname remotehostname:*
    - Copies the entire directory hierarchy of *dirname* to the home directory on the remote machine. Links (ie shortcuts) will cause problems.
  - **-v** debugging/verbose printing
  - **-p** preserve modification time, access times and modes

vi editor

- **`vi`** *`filename`* - opens (creates) a file using vi
- **`vi -R`** *`filename`* - opens a file using vi in read-only mode
- **`view`** *`filename`* - same as **`vi -R`** *`filename`*
- Two modes
  - insert mode
    - for typing in text
    - all keystrokes are interpreted as text
    - **`i`** one of the commands that initiates insert mode
  - command mode
    - for navigating the file and editing
    - all keystrokes are interpreted as commands
    - **`Esc`** returns the user to command mode

vi editor

```
netid@grace1 ~]$ vi filename
```

```
~
~
~
"filename"
```

starts in command mode

Typing :set showmode while in command mode will display in the lower right hand corner what mode you are in

vi commands

- To exit a file or save
  - **`ZZ`** or **`:wq`** or **`:x`** - save the file and exit
  - **`:w`** *filename* - save the file with the name filename
  - **`:w!`** - force save
  - **`:q`** or **`:q!`** -quit without saving
    - **`:q`** quits a file when there have been no changes
    - **`:q!`** quits the file regardless of changes

- Moving around in the file
  - **`h`**, **`l`** (or **`space`**), **`j`** and **`k`** - left, right, down and up
  - **`G`** Move to end of file
  - **`^f`** (^ = Ctrl-key) Scroll down a full screen
  - **`^b`** Scroll up a full screen
  - **`0`** (zero) Move to start of current line
  - ***n*`G`** Go to line *n*
  - **`w`** move forward one word
  - **`b`** move back one word
  - **`e`** move to the end of the word

vi commands

- Commands that take you into insert mode
  - **`i`** insert text to the left of the cursor
  - **`I`** inserts text at the beginning of the line
  - **`a`** insert text to the right of the cursor
  - **`A`** insert text at the end of the line
  - **`o`** open a line below the cursor
  - **`O`** open a line above the cursor
  - **`R`** overwrite text to the right of the cursor
  - **`cw`** changes a word to the text that you type it - the cursor must be at the beginning of the word

vi commands

- Editor commands that keep you in command mode
  - **x** deletes a character (the character the cursor is on)
  - **dd** deletes a line (the line the cursor is on)
  - *n***dd**deletes *n* lines
  - **dw** deletes a word
  - **dG** deletes to the end of the file
  - **D** deletes to the end of the line
  - **r***a* replaces current character with *a* (a = character, number, etc.)
  - **u** undo last command (only 1 undo on most unix machines. Most new versions of vi (vim) have multiple undo and redo (Ctrl-r) capability)
  - *n***yy**yank *n* (*n* is a number) lines to memory
  - **p** put the yanked lines below the cursor
  - **P** put the yanked lines above the cursor

vi commands

- Miscellaneous commands
  - **/*name*** search forward for name
  - **?*name*** search backward for name
  - **:1,$ s/*pattern1*/*pattern2*/g**
    - from line 1 to the bottom find and substitute *pattern1* for *pattern2*
    - you could also use **:% s/*pattern1*/*pattern2*/g**
      - **%** and **1,$** mean the entire file
    - the **g** means that all occurrences of *pattern1* will be substituted in a line and not just the first one
  - **:e *filename*** exits to the file *filename*
  - **ma** marks that line and stores the position in the variable a
  - **:'a,. y x** yanks the lines between the mark a and where the cursor is (.) and stores it in the variable x
  - **:pu x** puts the lines stored in x into the file where the cursor is
  - **:r** *filename* insert the file filename into the current file.
  - **:set all** lists all of the settings
  - **:set number** displays line numbers

Need Help? Contact the HPRC Helpdesk

Website: hprc.tamu.edu
Email: help@hprc.tamu.edu
Phone: (979) 845-0219

Help us, help you -- we need more info

- Which Cluster (Terra, Grace)
- NetID (NOT your UIN)
- Job id(s) if any
- Location of your jobfile, input/output files
- Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem