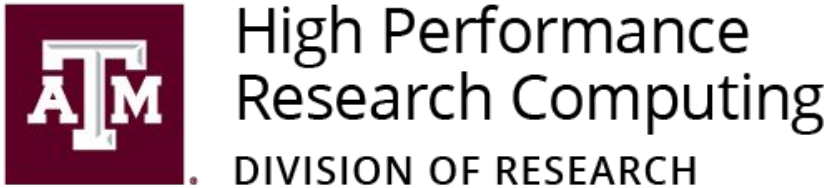
Working on HPC Systems



Wes Brashear

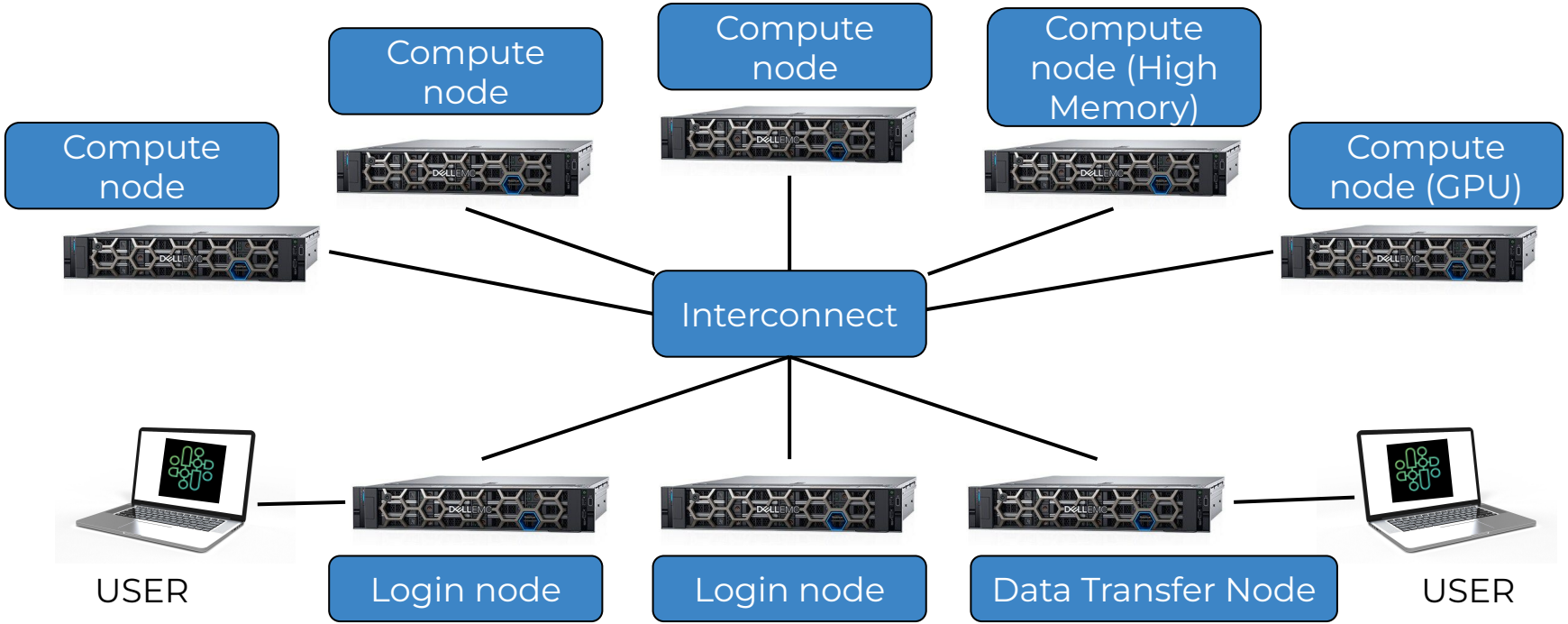
Wednesday, 26 February 2025

PACES Research Training Workshop

Outline

- HPC Architecture
- Bash Command Syntax
- Managing Directories and Files
- Useful Commands and Tools
- Job Orchestration
  - Submitting Jobs
  - Interactive sessions

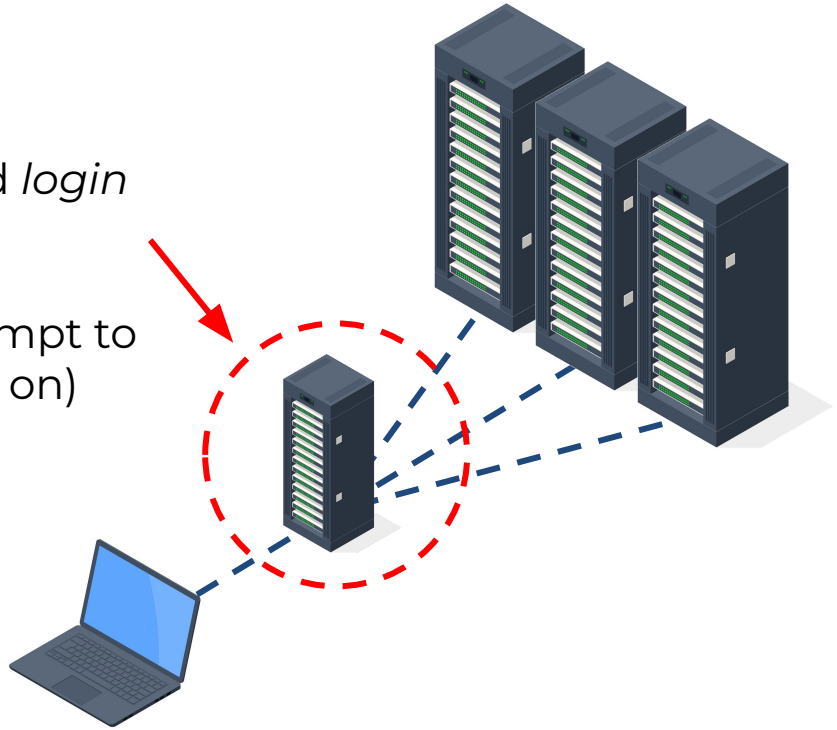
HPC Architecture



Shell Access via the Portal: Logging In

When you first log in, you're on a dedicated *login node*.

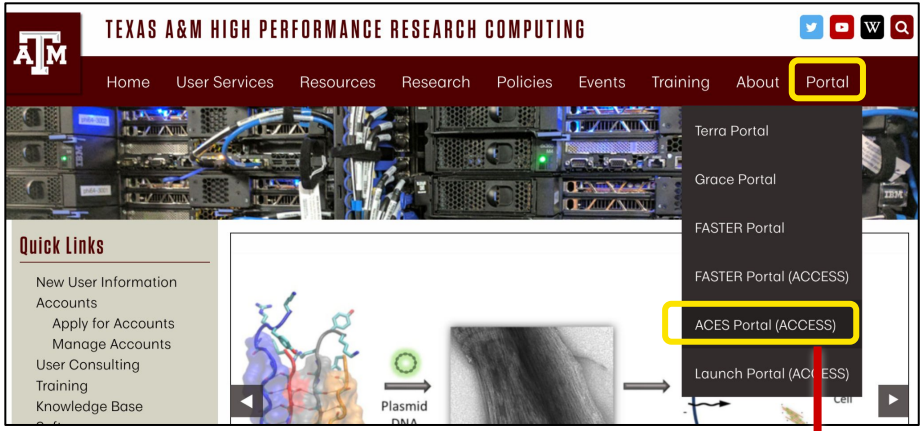
(check your shell prompt to see which one you're on)

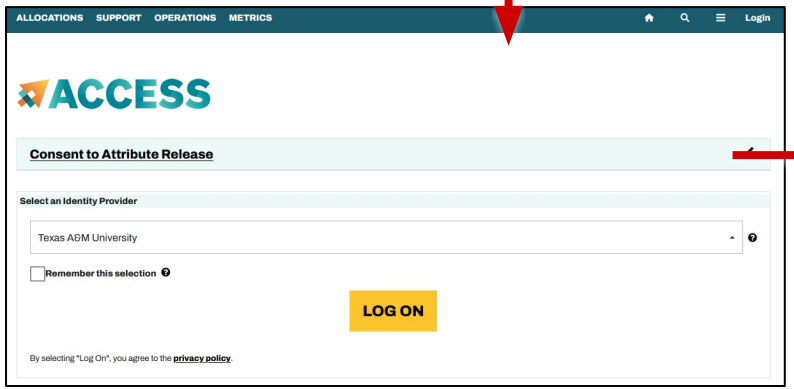


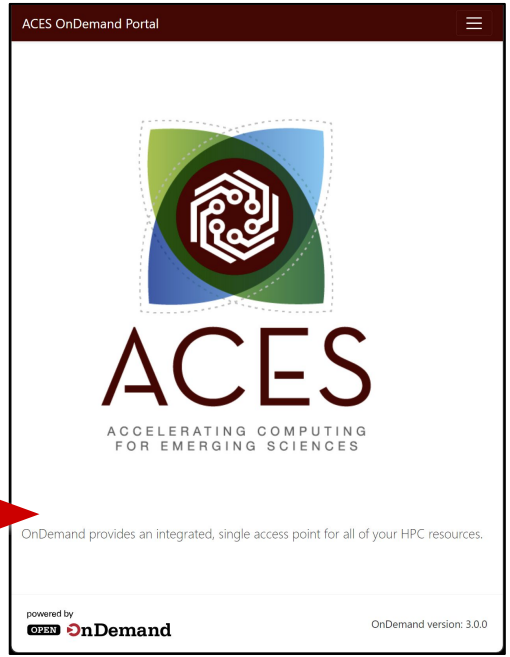
Login nodes are not for running big processes!
There are rules:

- No processes longer than 1 hr
- Sessions idle for 1 hr will be killed
- Don't use more than 8 cores
- Don't use "`sudo`"

ACES Access - Portal

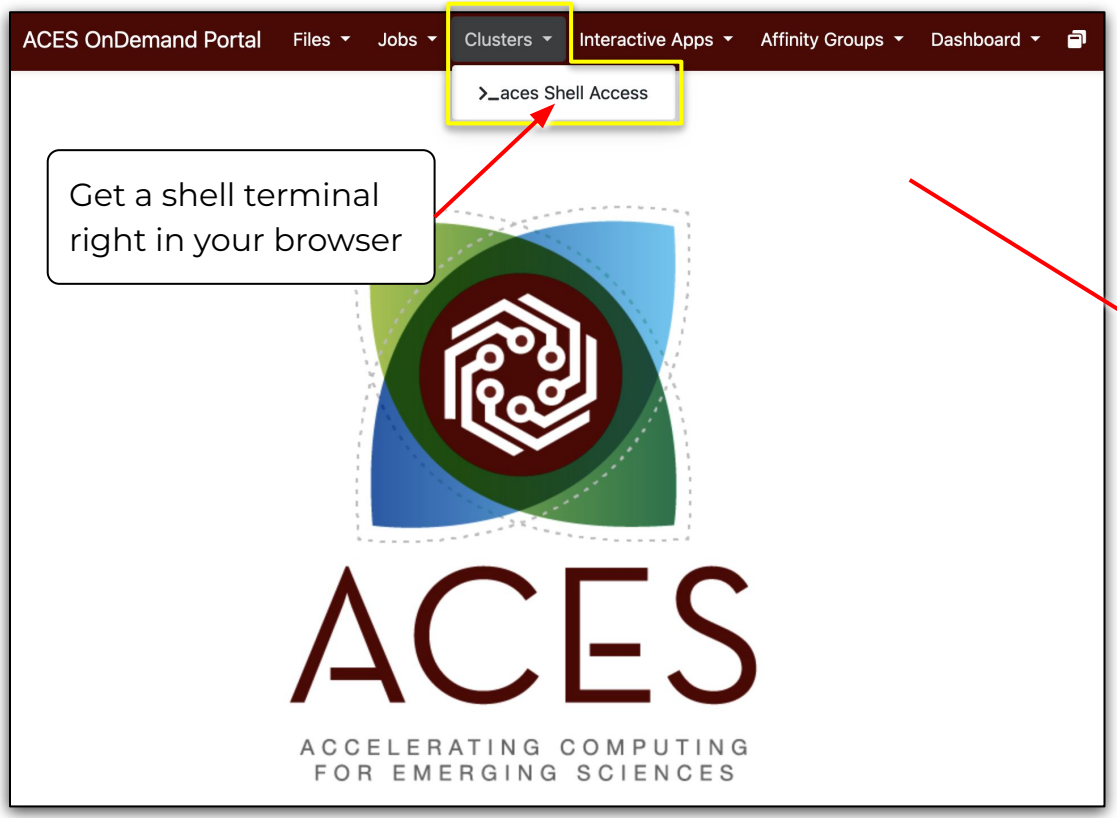






- Navigate from HPRC homepage
- Login via ACCESS
- Open-OnDemand Portal

Shell Access via the Portal

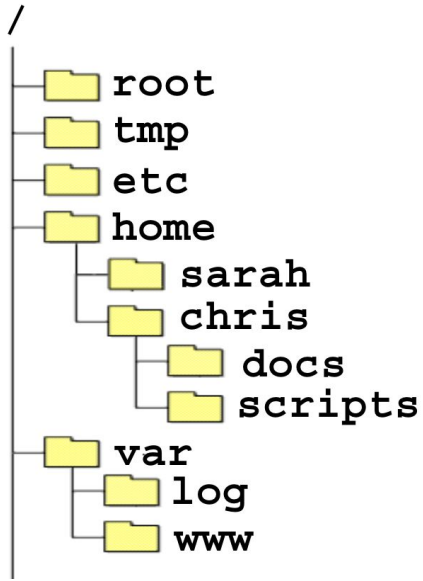


Bash Command Syntax

When a command is typed at the prompt, the Shell processes the command and sends it to the Linux kernel.

- Linux commands are case-sensitive
- Command line structure: Command [options] [arguments]
  - Example: [*netid*@grace1 ~] : **`ls -al /home/user/dir_name`**
    - [*netid*@grace ~] : is the prompt
    - **`ls`** is a command
      - list all the files in the current directory
    - **`-al`** are options
      - options typically starts with dash, changes the way commands work
    - **`/home/user/dir_name`** is an argument
      - arguments - input given to a command to process

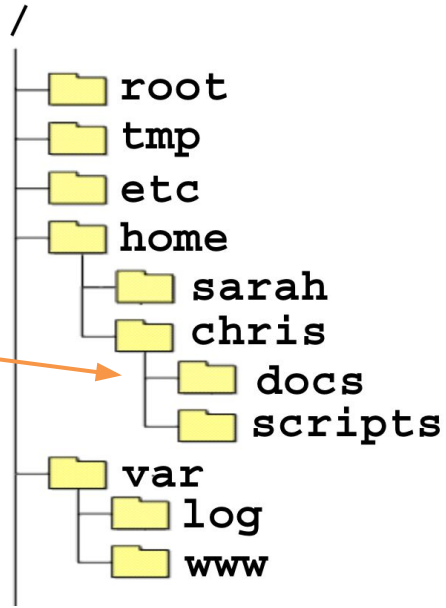
File Hierarchy Structure



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

Navigating the File System

- Most Linux file systems are case-sensitive.
- **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



Listing 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 all files including hidden files
    - -l print long listing
    - -C list entries by columns
    - -F print a special character after special files
    - to find all possible flags, use the command: man ls
- **tree** - recursive directory listing

File & Directory Names

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

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

- 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

Managing Files & Directories: mkdir

- 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 (creates the directory tmp in /home/*netid*)

Managing Files and Directories: mv

- Rename a directory
  - **mv** *olddirname newdirname*
- 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 and Directories: cp

- 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 and Directories: cp

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

Managing Files and Directories: rm

- 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 (we'll talk about these later). rm -r * will remove everything from that directory and down the hierarchy!

Exercise: Directories & Files

- Change to your home directory
- Print your current working directory
- List contents of the current directory including hidden files
- Make two directories named **temp1** and **temp2** in your current directory
- Show the current directory hierarchy using the **tree** command

Solution: Directories & Files

- Change to your home directory

```
cd
```

- Print your current working directory

```
pwd
```

- List contents of the current directory including hidden files

```
ls -a
```

- Make two directories named **temp1** and **temp2** in your current directory

```
mkdir temp1
mkdir temp2
```

- Show the current directory hierarchy using the **tree** command

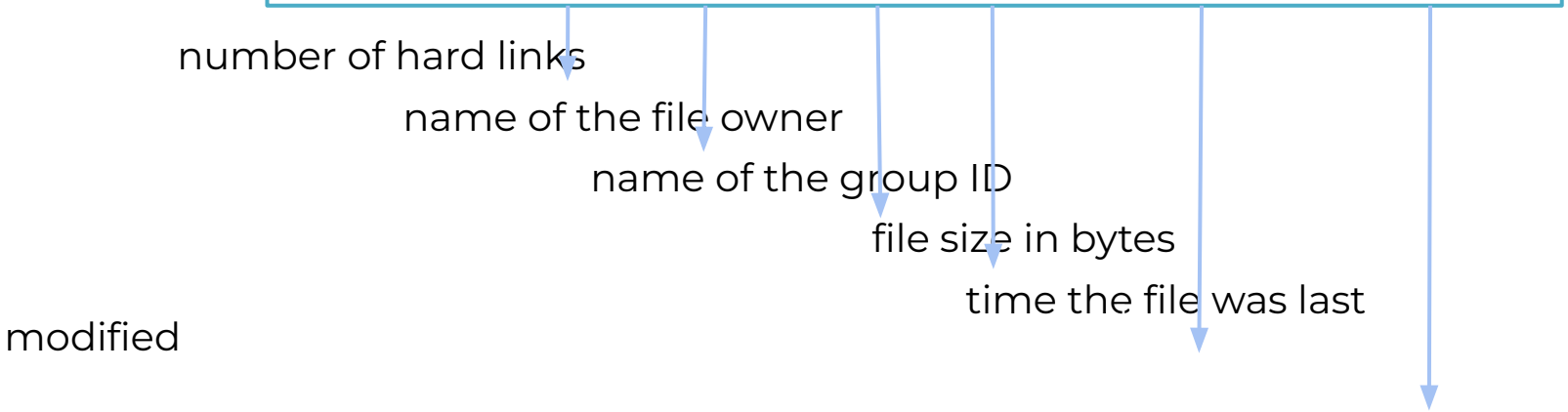
```
tree
```

File Attributes

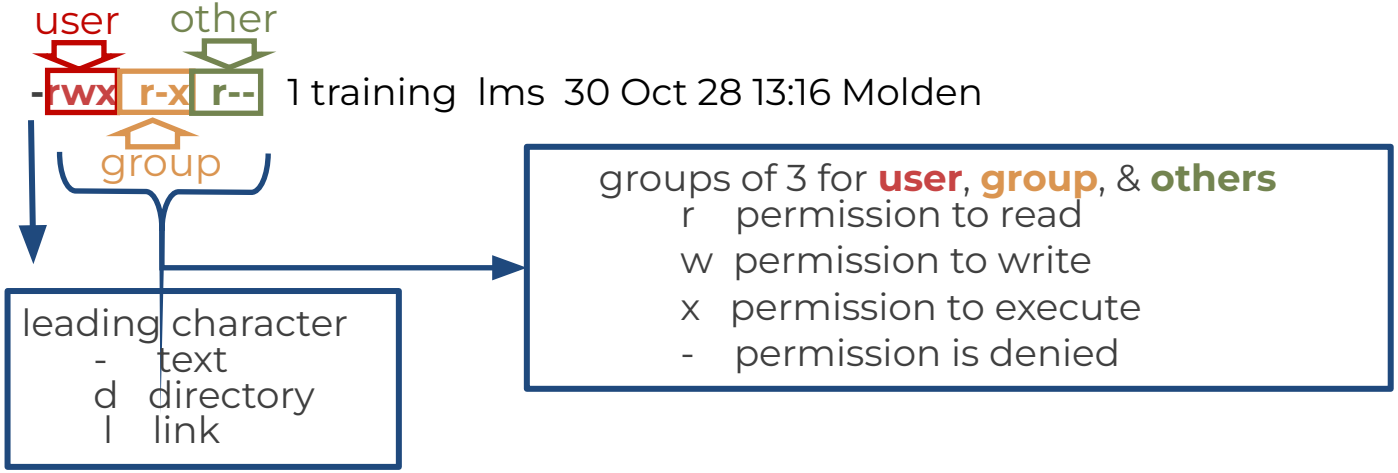
`ls -l` lists the files in the dir in **l**ong format

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

Example output: **-rwxr-xr-- 1 training ims 30 Oct 28 13:16 Molden**



File Attributes



Example:
-rwxr-xr-- 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*)
  - removes read permission for group and others
- **chmod -R a+rx** *dirname*
  - gives 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

Displaying the Contents of a File

Printing ASCII (text) file contents to the screen

- **less** *filename*
- **more** *filename*
- **cat** *filename*
- **cat -A** filename
  - shows 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*
  - Displays the last 10 lines of a file and waits for new lines, ctrl-c (^c) to exit.

Useful Commands & Tools

Searching File Contents

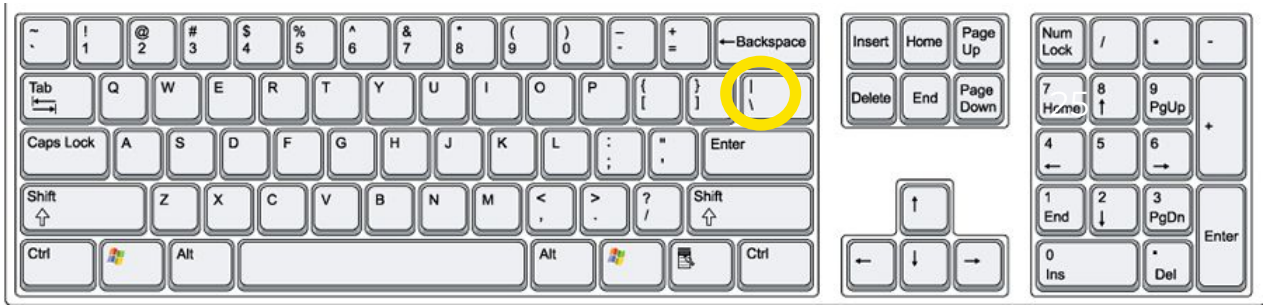
grep *search-pattern filename* - searches the file *filename* for the pattern *search-pattern* and shows the results on the screen (prints the results to standard out).

- **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 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*' *filename*

- 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.



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, !, ↑, ↓

- **history**
  - The history command will list your last n commands (n = integer).
    !! repeats your last command
    !*n* repeats the nth command
    !*name* repeats the last command that started with name

- You can use the up (↑) and down (↓) arrow keys to scroll through previous commands
- Examples:
  - **history | grep** *wget*
    search history commands that contains wget
  - **history | tail**
    see the last 10 commands

Using Tab for Autocompletion

Tab will try to complete the rest of the file/directory name you are typing

Example:

Type the first few characters of the file name

```
ls my
```

Then hit the **tab key** to autocomplete the file name

```
ls my_favorite_foods.txt
```

Then hit enter to see the command results

If the tab key did not complete the file name then either the file does not exist or there are two or more files that begin with the same characters in which case you need to hit tab twice then type a few more characters and hit tab again to complete.

Slurm SBATCH Parameters

Slurm Job Script Example

```
#!/bin/bash
#SBATCH --job-name=spades       # keep job name short with no spaces
#SBATCH --time=1-00:00:00       # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1               # request 1 node
#SBATCH --ntasks-per-node=1     # request 1 task (command) per node
#SBATCH --cpus-per-task=1       # request 1 cpu (core, thread) per task
#SBATCH --mem=5G                # request 5GB total memory per node
#SBATCH --output=stdout.%x.%j   # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j    # save stdout to a file with job name and JobID appended to file name

# unload any modules to start with a clean environment
module purge
# load software modules
module load GCC/11.3.0  SPAdes/3.15.5
# run commands
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

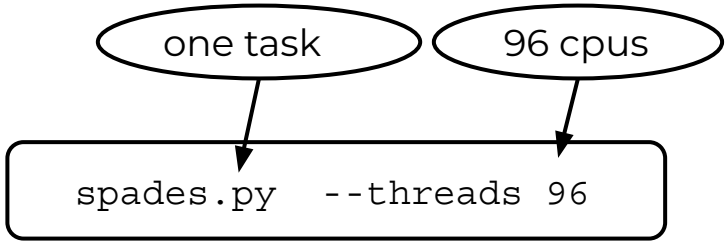
- Always include the first line exactly as it is; no trailing spaces or comments.
- Slurm job parameters begin with **`#SBATCH`** and you can add comments afterwards as above.
- Name the job script whatever you like, but be consistent to make it easier to search for job scripts.
  - **`my_job_script.job`**
  - **`my_job_script.sbatch`**
  - **`run_program_project.sh`**
  - **`job_program_project.slurm`**

Commonly Used Slurm SBATCH Parameters

- **`--nodes`**
  - number of nodes to use where a node is one computer unit of many in an HPC cluster
    - `--nodes=1` `# request 1 node`
  - used for multi-node jobs
    - `--nodes=10`
  - if number of cpus per node is not specified then defaults to 1 cpu
  - can be used with --ntasks or --ntasks-per-node

either `--ntasks`, `--ntasks-per-node` or `--nodes` needs to be provided.

- **`--ntasks`**
  - a task can be considered a command such as blastn, bwa, script.py, etc.
  - `--ntasks=1` `# total tasks across all nodes where each task is scheduled a max of 1 cpu`
  - when using --ntasks > 1 without --nodes=1, the job might be scheduled on multiple compute nodes
- **`--ntasks-per-node`**
  - use together with --cpus-per-task
  - `--ntasks-per-node=1`
- **`--cpus-per-task`**
  - number of CPUs (cores) for each task (command)
  - `--cpus-per-task=96`



Commonly Used Slurm SBATCH Parameters

- **`--time`**
  - max runtime for job (*required*); format: days-hours:minutes:seconds (days- is optional)
  - **`--time=24:00:00`** **`# set max runtime 24 hours (same as --time=1-00:00:00)`**
  - **`--time=7-00:00:00`** **`# set max runtime 7 days`**
- **`--mem`**
  - total memory for each node (*required*)
  - **`--mem=488G`** **`# request 488GB total memory (max available for 512gb nodes)`**
- **`--job-name`**
  - set the job name; keep it short and concise without spaces (*optional but highly recommended*)
  - **`--job-name=myjob`**
- **`--output`**
  - save all stdout to a specified file (*optional but highly recommended for debugging*)
  - **`--output=stdout.%x.%j`** **`# saves stdout to a file named stdout.jobname.JobID`**
- **`--error`**
  - save all stderr to a specified file (*optional but highly recommended for debugging*)
  - **`--error=stderr.%x.%j`** **`# saves stderr to a file named stderr.jobname.JobID`**
  - use just --output to save stdout and stderr to the same output file: **`--output=output.%x.%j.log`**
- **`--partition`**
  - specify a partition (queue) to use (*optional, use as needed*)
  - partition is automatically assigned to cpu so you don't need --partition unless you want to use accelerators.
    - need to specify **`--partition`** parameter to use gpu, bittware, memverge, nextsilicon, pvc

Commonly Used Optional Slurm Parameters

- `--gres`
  - used for requesting 1 or more GPUs; use GPU type in lowercase
  - use **`gpuavail`** command to see number of GPUs per compute node
  - **`--gres=gpu:h100:1`** `# request 1 H100 GPU; use replace :1 with :2 for two GPUs, etc`
  - **`--partition=gpu`** `# also include this line when requesting GPUs`
- `--account`
  - specify which account to use; use **`myproject`** to see your accounts
  - **`--account=ACCOUNTNUMBER`**
  - default account from **`myproject`** output is used if not specified
- `--mail-user`
  - **`--mail-user=myemail@myuniversity.edu`**
- `--mail-type`
  - send email per job event: BEGIN, END, FAIL, ALL
  - **`--mail-type=ALL`**
- `--dependency`
  - schedule a job to start after a previous job successfully completes
  - **`--dependency=afterok:JobID`**
    - get the JobID of the previous job with **`squeue --me`**

Submitting Slurm Jobs

- A job script is a text file of Unix commands with **#SBATCH** parameters.
- **#SBATCH** parameters provide resource configuration request values.
  - time, memory, nodes, cpus, output files, …
- Jobs can be submitted using a job script or directly on the command line.
  - start time depends on available resources
- Submit the job using sbatch command with the job script name.
  - Your job script provides a record of commands used for an analysis.
  - ```
    sbatch my_job_script.job
    ```
- Submit command on the command line by specifying all necessary parameters.
  - ```
    sbatch -t 01:00:00 -n 1 -J myjob --mem 5G -o stdout.%j commands.sh
    ```
- You can start an interactive job on the command line using the **srun** command instead of **sbatch**. Your srun job ends when you exit the terminal.
  - Do not to use more than the requested memory and CPUs when your **srun** job starts.
  - ```
    srun --time=04:00:00 --mem=5G --ntasks=1 --cpus-per-task=1 --pty bash
    ```

slurm.schedmd.com/sbatch.html
    ```