HIGH PERFORMANCE COMPUTING FROM TERMINAL WINDOW

Ahmad Sheikhzada & Gladys Andino

UVA Research Computing

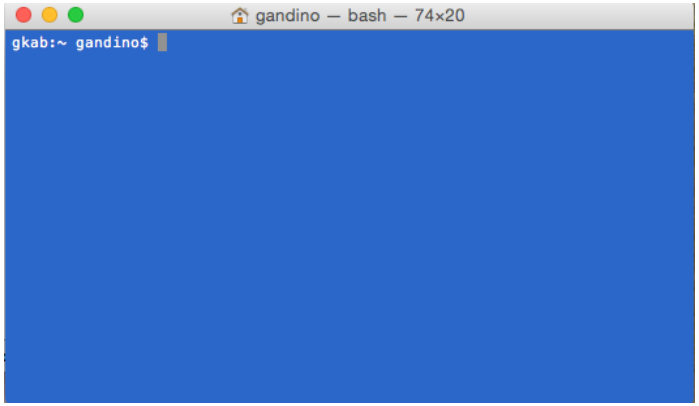
OUTLINE

- UNIX
- Rivanna
- Terminal emulation/logging in
- Basic Unix commands
- Directories and files + exercise
- File Manipulation + exercise
- Wildcards, Streams, Pipes, Executables,...
- Modules - How to load modules
- Shell & Slurm scripts + exercise

OVERVIEW - UNIX

- **UNIX**: is a text-oriented operating system originally developed at Bell Labs during 1960s.
  - Linux
    - Dominate OS used at HPC facilities, internet servers, majority of financial trades worldwide, billion Android devices.
  - Mac OSX graphical operating system.
  - GUI is available, but mostly through shell.

- Navigation
- Manipulation of files and directories
  - Examine files
  - Edit files
  - Execute files
  - Transfer files



OVERVIEW - UNIX

UNIX:

- **Kernel** is at the heart of the OS.
  - Allocation of time and memory to programs/applications
  - Handling the filesystem and communications in response to system calls
  - …
- **Shell** is a program that interprets commands and acts as an interface between the user and the kernel.
- **Programs/Applications**

OVERVIEW - SHELL

- There are several shells available. In most newer systems the default is `bash` (Bourne again shell). MacOS has switched recently to `zsh`.
- You will have a customizable **prompt** which indicates that the shell is ready to accept commands. On Rivanna the default is `-bash-4.2$`
- To find out your shell, at the prompt type `echo $shell`
- Unix in general and the shell in particular is **case-sensitive**.

OVERVIEW – SHELL CONT.

- The syntax of commands is not completely standardized but in general is

```
cmd -o --options filename
```

- The pattern is a two or three-letter abbreviation, single-letter options preceded by one hyphen, multiple-letter options with two hyphens, and arguments at the end.
- Example:

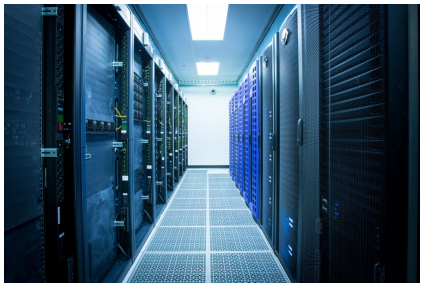
```
rm myfile
```

1. Shell searches for the file that contains the program rm
2. Shell requests kernel through system calls to execute rm on myfile
3. Shell then returns the UNIX prompt to the user, indicating that it is waiting for further commands.

OVERVIEW – SUPERCOMPUTER

Rivanna

Rivanna is the university's primary resource for high-performance computation. It provides a platform for computationally-intensive research across disciplines.



LOGGING IN

- Logging into a remote UNIX based system requires a client. The options for client depends on your OS.
- Command line access is based on the “SSH” or Secure Shell protocol
    - Encrypted
    - Used on most UNIX systems

LOGGING IN

- OOD
  - https://rivanna-portal.hpc.virginia.edu/
- FastX
  - https://rivanna-desktop.hpc.virginia.edu/
  - OOD > Interactive Apps > FastX web
- MobaXterm (Windows)
  - https://www.rc.virginia.edu/userinfo/rivanna/logintools/mobaxterm/
- Terminal (Mac, Linux)
  - `ssh -Y gka6a@rivanna.hpc.virginia.edu`

UNIX COMMAND-LINE BASICS

SHELL NAVIGATION

- Let's run our first command

`$ pwd` *(print working directory)*

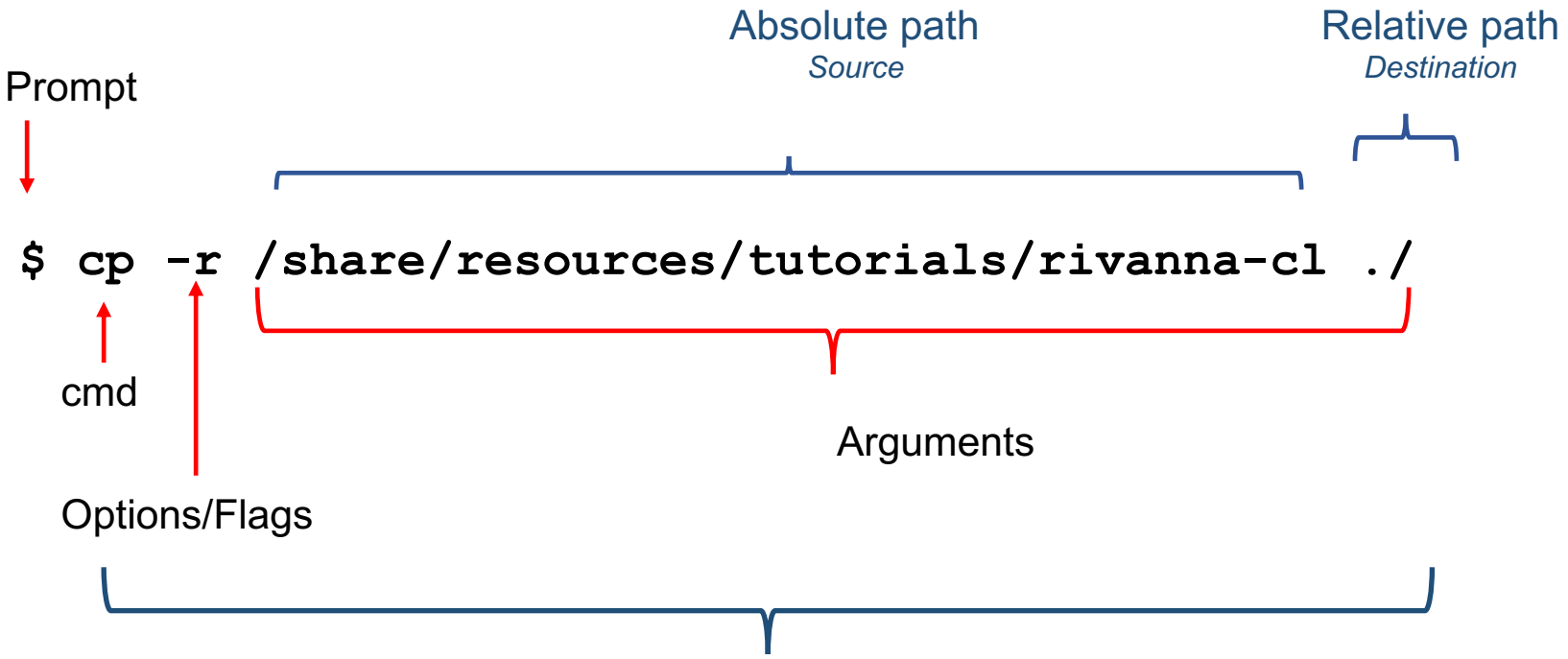
command: prints working directory

`/home/username` # username should be your UVA computing ID

Know where you are!

```
$ pwd
/home/gka6a
```

FIRST LOOK – Let's grab some files



Spaces separate these parts!

BASH HISTORY MECHANISM

- When using bash you may use its built-in history mechanism to save yourself some keystrokes.
  - Up arrow: scroll through the previous commands you have typed
  - Down arrow: if scrolled back, scroll to more recent commands
  - Left/right arrows: edit text on a line

MORE BASH GOODIES

- Tab completion

  `string<tab>` causes bash to expand string further as far as it has a unique name.

- Search for earlier command

  `control-r  <text>`

- Move to the beginning of the line

  `control-a`

- Move to the end of the line

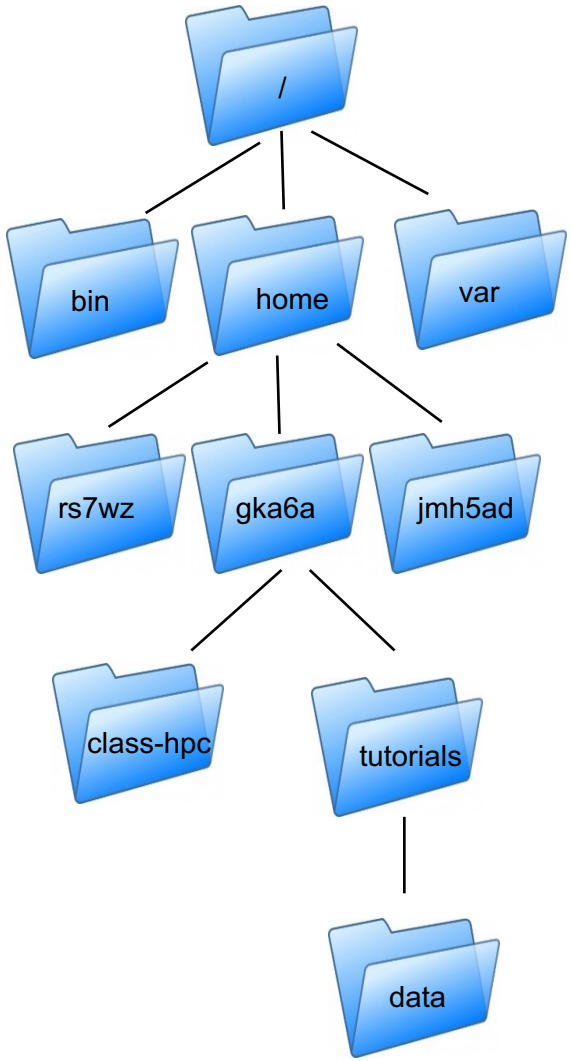
  `control-e`

- Clear the screen

  `clear or control-l`

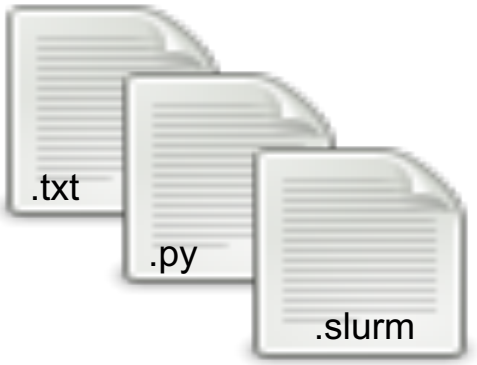
FILES AND PROCESSES

- Everything in UNIX is either a file or a process. Directories are special type of files.
- Files and directories are two important constructs in UNIX (and most operating systems).
- Contain your documents, images, code, programs, OS, etc.
- A “filesystem” is a collection of files and directories stored on a single physical device.
    - Often called “drives” in Windows



FILES AND DIRECTORIES

- Files store some sort of information
  - Two basic types of files:
    - Text (documents, code)
    - Binary (images, executables)



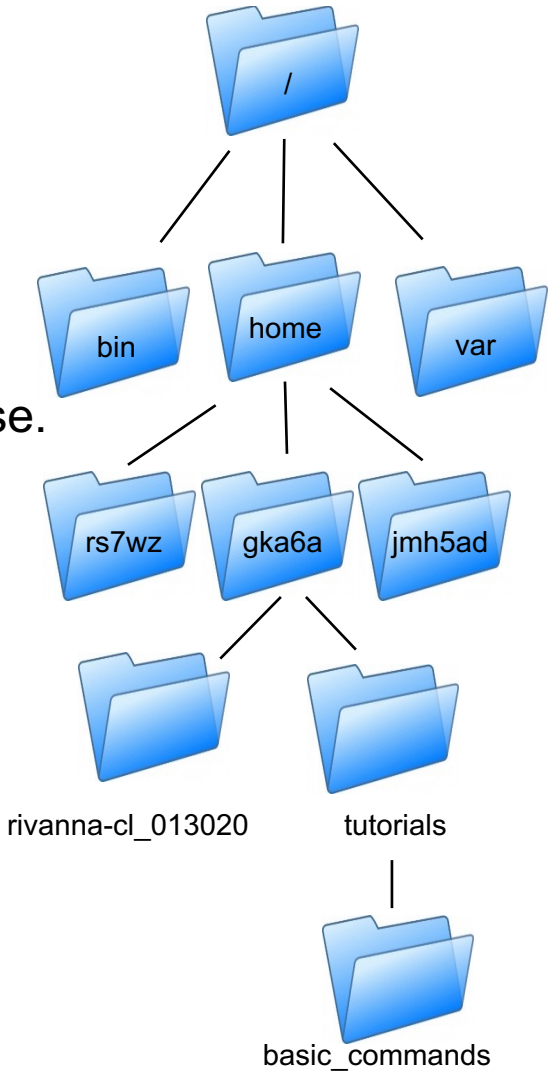
  - Unix doesn't pay attention to file extensions, but software might
- Directories are collections of files and directories
  - Analogous and interchangeable with "folders"

- Both files and directories have "metadata" associated with them
  - Name, timestamps, permissions

PATHS

- In UNIX all files and directories have a “path”
  - The “path” is the fullname of every file & directory
- At the top is the **root directory** that holds everything else. We refer to it using a slash character **/** on its own; this is the leading slash in `/home/gka6a`
- Examples:

```
/
/home/gka6a
/home/gka6a/rivanna-cl_013020
/home/gka6a/rivanna-cl_013020/basic_commands
```



PATHS

Example:

`/home/gka6a/rivanna-cl` **# This is an absolute path.**

The current directory can be represented by a period (`.`) Thus if we are in `/home/gka6a/rivanna-cl/basic_commands` we can type

```
gedit ./hello_world.sh &     # to edit or

./hello_world.sh             # to execute the file
```

PATHS - SUMMARY

- **Absolute paths**
  - The path to a file starting at the root of the system
  - Begins with "/" to denote the path starts at the root
  - Guaranteed to get you there

```
/home/gka6a/file.txt
/home/gka6a/files/file.txt
/home/gka6a/files/
/project/gka6a
```

- **Relative paths**
  - The path to a file starting at the current location
  - Indicate current directory with ".", parent directory as ".."
  - Can break if you start in the wrong place!

```
file.txt
./file.txt
files/file.txt
../gka6a/files/
../../project/
```

Basic Commands

- **`ls`** # to list files in current directory
- **`cd`** # to change directories
- **`mkdir`** # to create new directories
- **`less,more,head,tail,cat`** # to view the content of a file
- **`cp`** # to copy a file
- **`mv`** # to move or rename a file
- **`kill`** # to kill a process
- **`rm`** # to remove a file or directory
- **`grep`** # to capture patterns within a file
- **`man`** # to open the manual for a command
- **`clear`** # clear the content of terminal

LISTING FILES

`ls` command: list files in current directory

```
$ ls
basic_commands  data  protein  redirects  regex  scripts  Shakespeare
```

- `-l` long listing, includes file date and size
- `-a` displays all files (including dot(.) files).
- `-h` show file sizes in human readable terms
- `-t` show the newest files first
- `-r` reverse the order

```
$ ls
basic_commands  data  protein  scripts
$ ls -l
drwxr-xr-x 2 gka6a users 2560 Jan 29 13:20 basic_commands
drwxr-xr-x 2 gka6a users 2048 Jan 29 13:20 data
drwxr-xr-x 2 gka6a users 3584 Jan 29 13:20 protein
drwxr-xr-x 2 gka6a users  512 Jan 29 13:20 redirects
drwxr-xr-x 2 gka6a users  512 Jan 29 13:20 regex
drwxr-xr-x 3 gka6a users 5632 Jan 29 14:48 scripts
drwxr-xr-x 2 gka6a users 5120 Jan 29 13:20 shakespeare
$ ls -lah
drwxr-xr-x  9 gka6a users 4.0K Jan 29 15:20 .
drwx--s--x 42 gka6a users  29K Jan 29 14:32 ..
drwxr-xr-x  2 gka6a users 2.5K Jan 29 13:20 basic_commands
drwxr-xr-x  2 gka6a users 2.0K Jan 29 13:20 data
-rw-r--r--  1 gka6a users  11K Jan 29 13:20 .DS_Store
drwxr-xr-x  2 gka6a users 3.5K Jan 29 13:20 protein
drwxr-xr-x  2 gka6a users  512 Jan 29 13:20 redirects
drwxr-xr-x  2 gka6a users  512 Jan 29 13:20 regex
drwxr-xr-x  3 gka6a users 5.5K Jan 29 14:48 scripts
drwxr-xr-x  2 gka6a users 5.0K Jan 29 13:20 shakespeare
$ ls -laht
drwxr-xr-x  9 gka6a users 4.0K Jan 29 15:20 .
drwxr-xr-x  3 gka6a users 5.5K Jan 29 14:48 scripts
drwx--s--x 42 gka6a users  29K Jan 29 14:32 ..
drwxr-xr-x  2 gka6a users 2.0K Jan 29 13:20 data
drwxr-xr-x  2 gka6a users  512 Jan 29 13:20 redirects
drwxr-xr-x  2 gka6a users 5.0K Jan 29 13:20 shakespeare
drwxr-xr-x  2 gka6a users  512 Jan 29 13:20 regex
drwxr-xr-x  2 gka6a users 3.5K Jan 29 13:20 protein
drwxr-xr-x  2 gka6a users 2.5K Jan 29 13:20 basic_commands
-rw-r--r--  1 gka6a users  11K Jan 29 13:20 .DS_Store
```

CHANGING DIRECTORIES

- **cd** *(change directory)* is used to jump from one directory to another.

```
$ cd rivanna-cl/basic_commands
$ ls -lh
total 245M
-rw-r--r-- 1 gka6a users 2.1K Jan 29 13:20 intro_basic-unix.txt
-rwxr-xr-x 1 gka6a users 197M Jan 29 13:20 list_of_reads.txt
-rw-r--r-- 1 gka6a users  46M Jan 29 13:20 sequences.fasta
-rw-r--r-- 1 gka6a users 1.6M Jan 29 13:20 SP_R1.fastq
-rw-r--r-- 1 gka6a users 1.6M Jan 29 13:20 SP_R2.fastq
…
```

- **cd ..**

  Changes your present location to the parent directory.

- **cd**

  with no directory name puts you into your home directory.

- **cd ~**

  Changes to home directory

CREATING DIRECTORIES

- **`mkdir`** (*m*ake *d*irectory)

```
$ cd rivanna-cl_013020/basic_commands

$ mkdir NEW_DIRECTORY

$ ls -lht
drwxr-xr-x 2 gka6a users    0 Jan 30 00:12 NEW_DIRECTORY
-rw-r--r-- 1 gka6a users 1.6M Jan 29 13:20 SP_R2.fastq
-rw-r--r-- 1 gka6a users  46M Jan 29 13:20 sequences.fasta
-rw-r--r-- 1 gka6a users 1.6M Jan 29 13:20 SP_R1.fastq
-rwxr-xr-x 1 gka6a users 197M Jan 29 13:20 list_of_reads.txt
-rw-r--r-- 1 gka6a users 2.1K Jan 29 13:20 intro_basic-unix.txt
```

Look NEW_DIRECTORY should be empty!!

CREATING DIRECTORIES

- You can use absolute paths or relative paths for the directory name.

- Let's work these examples!

```
mkdir newcode
mkdir /home/gka6a/newcode/build
cd newcode
mkdir ../oldcode
mkdir src
mkdir ../oldcode/src
```

EXERCISE

- Start a terminal
- The example prompt is `-bash4.2$`
- Type after your prompt

```
echo $SHELL
pwd
mkdir test_dir
cd test_dir
ls ..
```

- What is the full path of your home directory?

CREATING & EDITING FILES

- **`gedit`**
- **`vi(vim)`**
  - To enter the insert mode press **`i`**
  - To enter the command mode press **`ESC`**
  - To save the file enter **`:w filename`**
  - To exit without save **`:q`**
- **`nano`**
  - Immediately start typing
  - To exit: **`control+X`**
  - Type the filename and press **`Enter`**
- vscode
  - **`module load code-server/4.16.1`**
  - **`code-server`**
  - Open the browser and copy the given url from terminal (http://127.0.0.1:8080/)

TRANSFER FILES

(https://www.rc.virginia.edu/userinfo/rivanna/logintools/cl-data-transfer/)

scp

- `scp l_SOURCE jus2yw@rivanna.hpc.virginia.edu:r_TARGET`
- `scp jus2yw@rivanna.hpc.virginia.edu:r_SOURCE l_TARGET`

rsync

- `rsync -av ldir/ jus2yw@rivanna.hpc.virginia.edu:rdir`
- `rsync my_file jus2yw@Rivanna.hpc.virginia.edu:/scratch/$USER`

sftp

- `sftp jus2yw@Rivanna.hpc.virginia.edu`
  - `Sftp> put myfile` # transfer from local to Rivanna
  - `Sftp> get myfile` # transfer from Rivanna to local

VIEWING CONTENTS OF A FILE

- **less** FILENAME

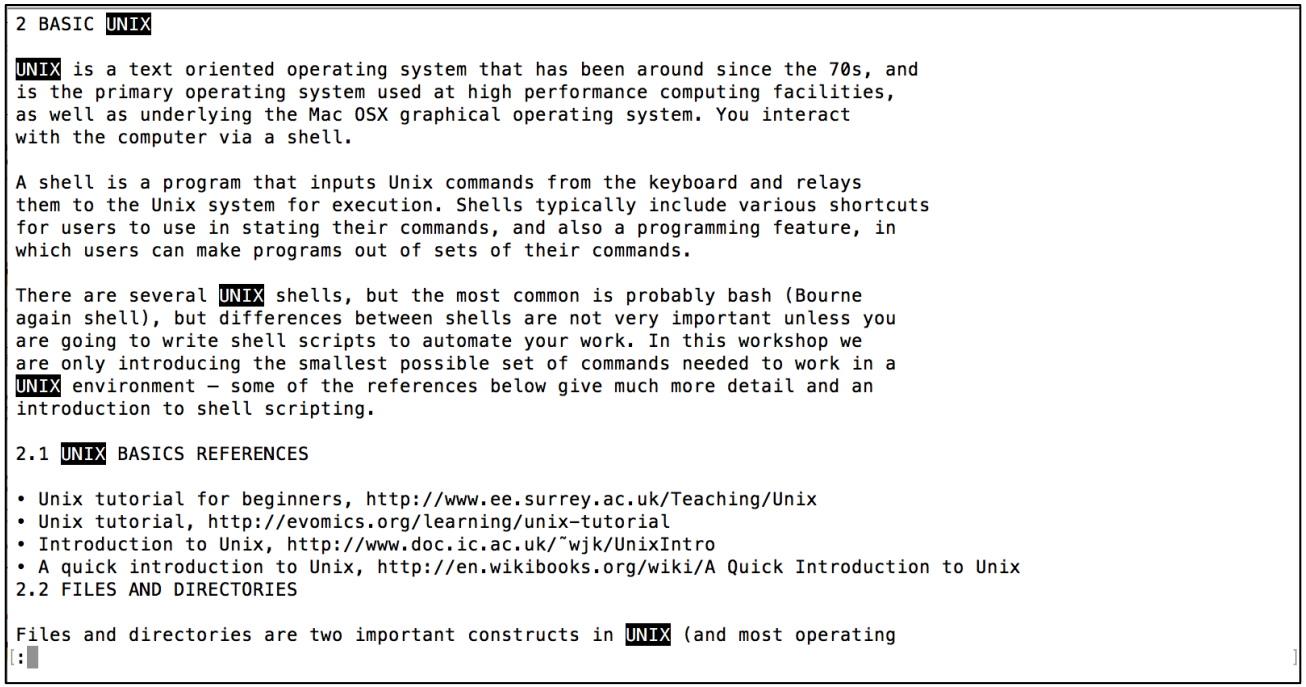
```
$ pwd
/home/gka6a/rivanna-cl_013020/basic_commands
$ less  intro_basic-unix.txt
```

- Displays file contents on the screen with line scrolling (to scroll you can use ‘arrow’ keys, ‘PgUp/PgDn’ keys, ‘space bar’ or ‘Enter’ key). Press “q” to exit.
- In most implementations you can search in the forward direction with /<pattern>

VIEWING CONTENTS OF A FILE

- /<WORD_PATTERN>
- /UNIX

SEARCHING FILE WHILE USING LESS



VIEWING CONTENTS OF A FILE

- **head** FILENAME
  - Displays only the starting lines of a file. The default is first ten lines. Use “-n” to specify number of lines.

  - **head intro_basic-unix.txt –n 5**

- **tail** FILENAME
  - Displays the last 10 lines.

  - **less intro_basic-unix.txt –n 5**

VIEWING CONTENTS OF A FILE

- **`grep`** is commonly used in UNIX to filter a file/input, line by line, against a pattern (e.g., to print each line of a file which contains a match for pattern).

```
grep [OPTIONS] PATTERN FILENAME
```

```
grep -i "Unix" intro_basic-unix.txt
```

- A handy trick for bioinformaticians

how many sequences are in a FASTA-formatted file? Each sequence record in a FASTA file has one line of description that always starts with ">" , followed by multiple lines of sequence itself. Each sequence record ends when the next line starting with ">" .

VIEWING CONTENTS OF A FILE

Let's look at the fasta file first

```
$ pwd
/home/gka6a/rivanna-cl/basic_commands

$ grep -c '>' sequences.fasta
?
```

Please note to include the quotes around the ">".

COPYING FILES

- **cp** (*copy*) is used to copy a file. When using this command, you must provide both source file and destination file.

	Source	*Destination*
`cp`	`oldfile`	`newfile`

- You can copy using relative paths

```
cp thisfile ../../place/else/thatfile
```

- Copy all files in a directory and its subdirectories:

```
cp -R thisdir thatdir
```

- Do not overwrite an existing file (noclobber):

```
cp -n oldfile newfile
```

- Ask before overwriting (-i and –n override each other):

```
cp -i oldfile newfile
```

MOVING A FILE OR DIRECTORY

- **mv** *(move)* is used to move or rename a file or directory.

Moving: **mv** FILE DIR

Renaming: **mv** OLDNAME NEWNAME

```
$ pwd
/home/gka6a/rivanna-cl/basic_commands

          OLDNAME          NEWNAME
```

```
$ mv SP_R1.list list_of_reads.txt
```

DELETING FILES AND DIRECTORIES

- **`rmdir`** (*re**m**ove* *dir*ectory) is used to delete empty directories from the system.

```
rmdir DIRECTORY
```

- **`rm`** (*re**m**ove*) is used to delete a file or a directory. “-r” option for recursive action.

```
rm -r  DIRECTORY
rm FILE
```

EXERCISE

- If not already in your `basic_commands` navigate there. Type after your prompt `bash$`

`cat > mynewfile`

`Ctrl + D` to save it as a new empty file.

Use nano or editor of your preference and type a line or two of text.

```
more mynewfile
ls
mv mynewfile the_file
cp the_file old_file
ls -l
```

YOUR BEST FRIEND

- Basic documentation for a command can be read from the shell with the **`man`** (manual) command.

```
man ls
```

- Alternatively:

```
whatis ls
```

HANDY COMMANDS

`whoami` # show my user ID

`which <executable>` # indicates the path to the executable specified

`wc (-l/-w/-m) <file>` # word/line/character count

`date` # Shows instantaneous date and time

`exit` # exit current shell (if login shell, this logs you off!)

WILDCARDS

- Strings of characters may be replaced with wildcards.
- The asterisk (*) can replace zero to unlimited characters (it does not replace a leading period).
- The question mark (?) replaces exactly one character. Wildcards enable you to work with files without needing to type multiple files with similar names.

```
ls *py
rm list?.sh
```

- BE CAREFUL when using wildcards with `rm`! Gone is gone!

On some systems there may be backups, or maybe not, and on your personal system you would need to set up backups and learn to retrieve files.

STANDARD STREAMS

- Each executable has associated with it, three I/O streams: **standard input**, **standard error**, and **standard output**.
- Normally these streams come from or go to your console (i.e. your shell).
- Most Unix commands read from standard input and/or write to standard output.
- They are often represented as **stdin**, **stderr**, and **stdout**.

STREAM REDIRECTION

- You can redirect standard input with <

```
mycode < params.txt
```

- Redirect standard output with >

```
ls -l > filelist
```

- Append with >>

```
cat file1 >> bigfile
```

- Redirection of standard error depends on the shell
- Bash:

```
make >& make.out
```

Redirects both stdout and stderr to make.out

PIPES

- One of the most powerful properties of Unix is that you can **pipe** the standard output of one command into the standard input of another.
- The pipe symbol | is above the backslash on most US keyboards.

Example:

```
grep "@H-148:116" SP_R1.fastq | head
```

grep searches for the pattern in the file and `head` looks at the first 10 lines of the **grep** output

RUNNING EXECUTABLES

- Executables are often called binaries, especially by Unix types and computer programmers. The terms are synonymous in most cases.
- If the executable is in your *search path,* you can simply type its name at the prompt.

```
gedit hello_world.slurm
```

here `gedit` is the name of the binary. Its actual location is `/usr/bin/gedit`, but `/usr/bin` is in the default search path.

- If it is not in your search path, you must type the path to the executable (can be absolute or relative)

```
./hello_world.slurm
```

Usually, current directory is not in your default search path for security reasons.

EXAMPLE

- In most cases, current working directory (.) is not in your default search path. To add it, type (for bash)

```
export PATH=$PATH:.
```

In this case it is essential to add the first `$PATH` or you will lose the default path set by the system.

PROCESS CONTROL

- Running from a command line:
- Processes can be running in the *foreground* (no prompt returned to the shell) or *background* (prompt available). To start in the background, add an ampersand (`&`) at the end,

```
./myexec -o myopt myfile &
```

- **`jobs`** # lists the running jobs with job index
- `control-z` (`ctrl-z` or `^z`): suspends the job
- **`bg %1`** # place the job number 1 into the background
- **`fg %4`** # place the job number 4 back to the foreground

KILLING PROCESSES

- `control-c` (`ctrl-c` or `^c`): kill the current running job (must be foregrounded).

- Find the process ID with **`ps`** then:

  `kill -9 <pid>`

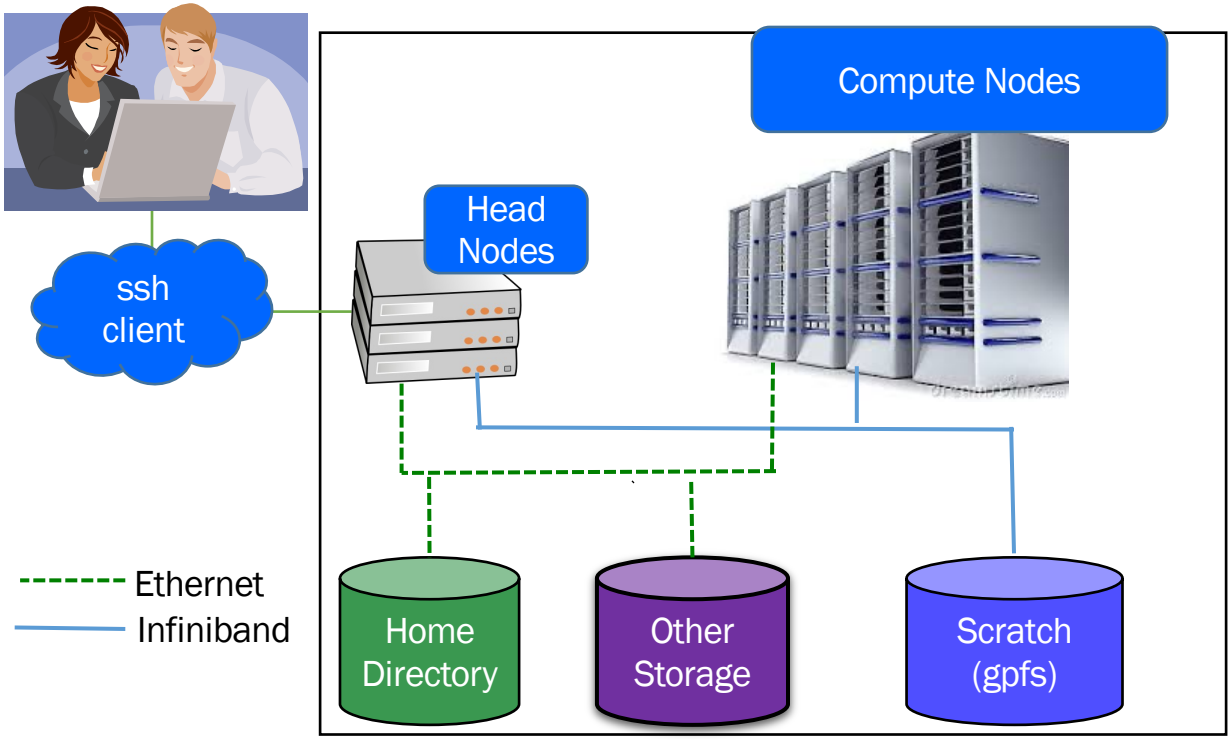
terminates with extreme prejudice.

- **`killall -9 <executable name>`**

  same as above.

RIVANNA

RIVANNA IN MORE DETAIL



YOUR HOME DIRECTORY

- The default home directory on Rivanna has 50GB of storage capacity, e.g., `/home/gka6a`
- Each user will have access to 10 TB of **temporary** storage, e.g., `/scratch/gka6a`
- The /home and /scratch directories are for personal use and not shareable with other users.

Important:
/scratch is **NOT permanent** storage and files that have not been accessed for more than **90 days** will be marked for deletion.

CHECKING YOUR STORAGE

- To see how much disk space, you have used in your home and scratch directories, open a terminal window and type **`hdquota`** at the command-line prompt:

```
$hdquota

Type    Location          Name                      Size Used Avail Use%
=============================================================================

home        /home         gka6a                      51G   12G   39G  24%
Project     /project      slurmtests                2.0P  1.9P  144T  93%
Project     /project      arcs                       16T   12T  3.8T  75%
Project     /project      rivanna_software          1.1T  4.2M  1.0T   1%
Project     /project      ds5559                     51G  3.7G   47G   8%
Value       /nv           vol174                    5.5T  1.2T  4.4T  21%
```

CHECKING YOUR ALLOCATION

- To see how many SUs you are available for running jobs, type at the command-line prompt: **`allocations`**

```
$ allocations

Allocations available to Gladys_Karina_Andino_Bautista (gka6a):

 * arcs_admin: less than 500 service-units remaining
 * ds5559: less than 25,000 service-units remaining
 * ga_bioinfo-test: less than 100,000 service-units remaining
 * hpc_build: less than 203,417 service-units remaining
 * rivanna-training: less than 20,000 service-units remaining


 for more information about a specific allocation, please run:
 'allocations -a <allocation name>'
```

RUNNING JOBS FROM SCRATCH

- We recommend that you run your jobs out of your `/scratch` directory for two reasons:
  - `/scratch` is on Weka filesystem (a storage system designed specifically for parallel access).
  - `/scratch` is connected to the compute nodes with Infiniband (a very fast network connection).

We also recommend that

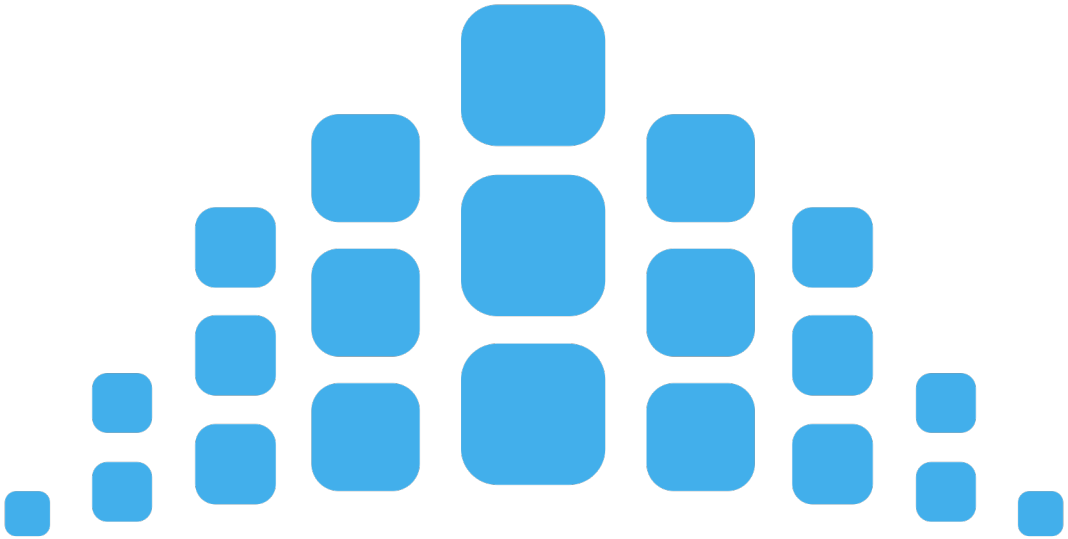
- You keep copies of your programs and data in more permanent locations (e.g., your home directory or leased storage such as /project or /value).
- After your jobs finish, you copy the results to more permanent storage).

THE MODULES ENVIRONMENT

- Not strictly a part of Unix
- Widely used by many HPC sites, including ours.
- Enables the user to set complex paths, environment variables, and so forth, by loading a module (running a script).
- The environment is set up automatically when you log in.
- Loaded modules only affect the shell in which the command is run.
- Modules required for a job must be loaded in the batch job script.

MODULES COMMANDS

- `module spider`
  - List all available packages (may be a lot!)
- `module spider <package>`
  - List all versions of <package>, if any
- `module spider <package>/<version>`
  - Describes how to load <package>/<version>. There may be prerequisite modules.
- `module list`
  - List modules loaded in current shell
- `module load <package>/[<version>]`
  - Load the module for (optionally) <version> of <package>
- `module unload <package>`
  - Delete the changes made by the <package> module
- `module purge`
  - Remove all module modifications to the environment
- `module swap <package>/<current> <package>/<newver>`
  - Exchange one version of a package for another



slurm

workload manager

https://www.rc.virginia.edu/userinfo/rivanna/slurm/

RESOURCE MANAGERS

- SLURM (Simple Linux Utility for Resource Management) is a **resource manager** (RM), also known as a *queueing system*.
- Resource managers are used to submit jobs to compute nodes from an access point generally called a *frontend*.
- Frontends are intended for editing, compiling, and very short test runs. Production jobs go to the compute nodes through the RM.

RESOURCE REQUESTS

- A **job** (aka **batch job**) is described through a special form of shell script which contains directives to the RM to request resources.
- Directives are pseudocomments that take a particular form. They are read by the RM to determine the resource requirement. The job is then placed into the queue.
- Once a resource becomes available the job is started on a master compute node. The master runs the job script, treating the directives as comments.

SLURM RESOURCE REQUESTS

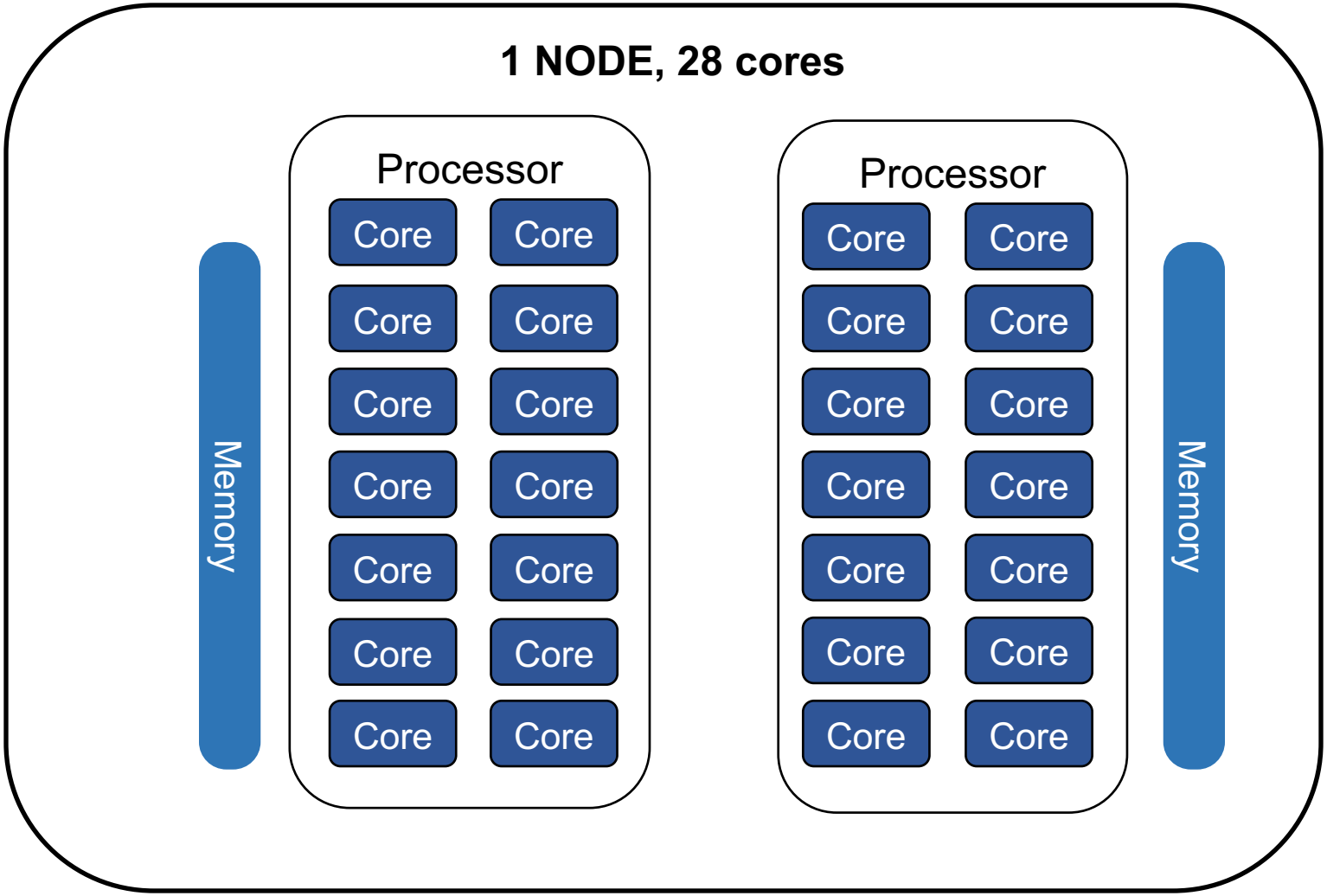
SLURM refers to queues as **partitions**. We do not have a default partition; each job must request one explicitly.

Queue Name	Purpose	Job Time Limit	Memory / Node	Cores / Node
standard	For jobs on a single compute node	7 days	384 GB	40
gpu	For jobs that can use general purpose GPU's (P100,V100,A100)	3 days	256 GB 384 GB 1 TB	28 40 128
parallel	For large parallel jobs on up to 50 nodes (<= 1500 CPU cores)	3 days	384 GB	40
largemem	For memory intensive jobs (<= 16 cores/node)	4 days	768 GB 1 TB	16
dev	To run jobs that are quick tests of code	1 hour	128 GB	4

SLURM RESOURCE REQUESTS

- Each master process corresponds to a task.
- By default, each task is assigned to one core.
- Resources are classified as
  - Core: often called "cpu" in directives but refers to each individual processing core
  - Node: a separate computing entity containing multiple CPU cores that can share the same memory.

SLURM RESOURCE REQUESTS



EXAMPLE

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem=32000 # mb total memory
#SBATCH --time=1-12:00:00
#SBATCH --partition=standard
#SBATCH --account=rivanna-training
./myexec < input > output
```

WHAT IT DOES

- The lines beginning with #SBATCH request
  - 1 node, 1 task, which by default uses a single core.
  - 32GB of memory (measured in MB).
  - 1 day and 12 hours of running time.
  - The standard partition (queue). A partition must be specified.
  - Account group rivanna-training
  - The job runs a serial executable myexec with input and output files redirected from standard input and standard output.
- We would call this script whatever name we choose, say myjob.slurm

SUBMITTING A JOB

- We use the sbatch command to submit the job:

  `sbatch myjob.slurm`

  The system returns a `JOBID`.

- We do not make the script executable. The system handles that.

```
udc-ba36-25$sbatch myjob.slurm
Submitted batch job 36805
```

CHECKING THE JOB

- Once we submit it, we can monitor active jobs with

```
squeue
squeue -u gka6a   # If I only want to see my jobs I can use
```

- Job status:
  - `PD` pending
  - `R` running
  - `CG` exiting

```
$ squeue -u gka6a

JOBID PARTITION     NAME     USER    ST     TIME  NODES  NODELIST(REASON)
36805  standard   myjob.sl  gka6a     R     1:45     1     udc-aw38-34-1
```

EXERCISE

- Change directory to scripts and run the slurm file `hello_world.slurm`

- What do you see when you run

```
squeue -u <yourID>
```

DELETING A JOB

- If you need to cancel a job use `scancel` with the job ID (you must always know the job ID)

```
$ scancel 36805 #jobID
```

COMMON SLURM DIRECTIVES

```
#SBATCH --nodes=<n>
#SBATCH --ntasks-per-node=<n>
#SBATCH --ntasks=<n>
#SBATCH --mem=<n> #in mb
#SBATCH --mem-per-cpu=<n>  #in mb
#SBATCH --cpus-per-task=<n> #cores
#SBATCH --time=<d-hh:mm:ss>
#SBATCH --account=<mygroup>
#SBATCH --partition=<p>
```

SINGLE-LETTER EQUIVALENTS

- Many SLURM directives have a single-letter equivalent which is used with a single hyphen and no equals symbol.

```
#SBATCH -N <n>   #nodes
#SBATCH -n <n>   #ntasks
#SBATCH -c <n>   #cpus-per-task
#SBATCH -t <d-hh:mm:ss> #time
#SBATCH -A <mygroup> #account
#SBATCH -p <p>
```

STANDARD STREAMS

- By default, SLURM redirects both standard output and standard error to `slurm-<jobid>.out`
- Change the name of this file:

```
#SBATCH --output=<filename>
```

- or

```
#SBATCH -o <filename>
```

- Separate standard error

```
#SBATCH --error=<filename>
```

- or

```
#SBATCH -e <filename>
```

INTERACTIVE JOBS

- Most HPC sites, including UVa's, restrict the memory and time allowed to processes on the frontend.
- The basic SLURM command to request interactive resources is **`salloc.`**
- However, it requires several options to work well so we have a local script called **`ijob`**

- **`ijob`** takes the same arguments as the SLURM commmand **`salloc`**

```
ijob -c 1 -A myalloc -t <time> --mem <memory
in MB> -p <partition> -J <jobname>
```

INTERACTIVE JOBS - EXAMPLES

- Jupyter-lab
  - `module load gcc jupyter_conda/.2020.11-py3.8`
  - `jupyter-lab &`

- Rstudio
  - `module load goolf/11.2.0_4.1.4 R/4.2.1 rstudio`
  - `rstudio &`

- MATLAB
  - `module load matlab/R2023a`
  - `matlab &`

NEED HELP?

Office Hours https://www.rc.virginia.edu/support/#office-hours

Research Computing staff host weekly office hours. Tell us about a project idea or talk to us about our high performance computing platforms, storage and services. We're here to help you solve your computational problems.

Examples of the type of support we can provide are:

- Data Transfer/Access
- Parallel Coding in Fortran, C, Python, R, Matlab, Mathematica
- Bioinformatics
- Computational Chemistry
- Software Installation and Containers
- Image Processing
- Writing Slurm Job Scripts
- Maximizing Job Efficiency
- Managing Computational Workflows

Beginning March 17, we are suspending our in-person office hours in the Physical & Life Sciences Building, Brown Library and the Health Sciences Library. We will be offering weekly office hours as online Zoom sessions instead until further notice.

Tuesdays 3:00-5:00pm
Join us via Zoom

Wednesdays 3:00-5:00pm
Join us via Zoom

Thursdays 10:00-12:00pm
Join us via Zoom

https://www.rc.virginia.edu/support/


UNIVERSITY of VIRGINIA
RESEARCH COMPUTING
CONSULTING
RIVANNA ACCESS
VISUALIZATION LAB
CODE OPTIMIZATION
SOFTWARE APPLICATIONS
OUTREACH AND EDUCATION
OPEN TO ALL UVA FACULTY, STAFF AND STUDENTS
RC.VIRGINIA.EDU

ADVANCED TOPICS

DOTFILES – CONFIGURATION FILES

- "Dotfiles" are files that describe resources to programs that look for them.
- They begin with a period or "dot" (hence the name).
- Hidden from `ls`, but `ls -a` shows them. Sometimes `ls` is aliased to `ls -a`.
- Bash has two `.bash_profile` and `.bashrc`
  - if no `.bash_profile` is present it will read `.profile`
- `.bash_profile` is sourced only for login shell

ENVIRONMENT VARIABLES

- The search path is the set of directories that the operating system will search when you type the name of an executable. To see it, type

```
printenv PATH
```

- `PATH` is an *environment variable*. Environment variables describe something about your working environment. Some of them you can set or modify; others are set by the system.
- `printenv`

  Prints all environment variables currently set.
- `export VAR`

  Allows variable to be passed to child shells
- Bash only, sets and exports in one line

```
export VAR=value
```

SLURM ENVIRONMENT VARIABLES

- SLURM provides several environment variables that acquire values from the system.

- Do not attempt to assign to any variable beginning with SLURM_

- Directory from which job was submitted `SLURM_SUBMIT_DIR`
  - Default is to cd to this directory before starting job

- List of nodes `SLURM_JOB_NODELIST`
- Job ID `SLURM_JOB_ID`
- Number of task `SLURM_NTASKS`
- Task per node (if set in script) `SLURM_NTASKS_PER_NODE`
- Number of cores per task (if set in script) `SLURM_CPUS_PER_TASK`

JOB ARRAYS

- Many similar jobs can be submitted through job arrays.
- Must be a batch job.
- Submit with `--array=<range>` option

```
sbatch --array=0-30 myjobs.sh
```

- An increment can be provided

```
sbatch --array=1-7:2 myjobs.sh
```

  - This will number them 1, 3, 5, 7
- Or provide a list

```
sbatch --array=1,3,4,5,7,9 myjobs.sh
```

- Each job will be provided an environment variable **`SLURM_ARRAY_JOB_ID`**
- And each task will be assigned **`SLURM_ARRAY_TASK_ID`** based on the numbers in the specified range or list.

ARRAY SCRIPT

- Job arrays should be named (any job can be, job arrays should be named).

```
#SBATCH --job-name=<name>
```

- or

```
#SBATCH -J <name>
```

- All subjobs will use the same global resource requests.
- A variable `%A` represents the overall `SLURM_ARRAY_JOB_ID` and `%a` represents `SLURM_ARRAY_TASK_ID` in the `-o` and `-e` directives.

FILE SPECIFICATIONS

- It would be prudent to separate stdout and stderror

```
#SBATCH -o myjobs%A%a.out
#SBATCH -e myjobs%A%a.err
```

- Prepare files with appropriate names, e.g.

```
myinput.0.in, myinput.1.in, …myinput.30.in
```

- Invoke your program with a line such as
- `./myexec myinput.${SLURM_ARRAY_TASK_ID}.in`

EXAMPLE SCRIPT

```
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 04:00:00
#SBATCH -J montecarlo
#SBATCH -A rivanna-training
#SBATCH -p standard
#SBATCH -o output%A.%a
#SBATCH -e error%A.%a
./mccode < input${SLURM_ARRAY_TASK_ID}.dat
```