Intro to HPC

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

HPC Job Submission

Daniel Trahan
Daniel.Trahan@Colorado.EDU
rc-help@colorado.edu
https://www.colorado.edu/rc

Slides available for download at
https://github.com/ResearchComputing/RMACC2021_Intro_to_HPC

Adapted from presentations by RC members Andrew Monaghan, Aaron Holt and John Blaas: 1, 2, 3, 4.

Outline

- Part 1: Intro to Linux
  - Linux Overview
  - Shells and environments
  - Commands
  - Files, Directories, Filesystems
- Part 2: Job Submission
  - General Info
  - Simple batch jobs
  - Running programs, MPI
  - Interactive jobs

Part 1: Linux

Linux Overview

- Part of the Unix-like family of operating systems.
- Started in early ‘90s by Linus Torvalds.
- Typically refers only to the kernel with software from the GNU project and elsewhere layered on top to form a complete OS. Most is open source.
- Several distributions are available; from enterprise-grade, like RHEL or SUSE, to more consumer-focused, like Ubuntu.
- Runs on everything from embedded systems to supercomputers.

Why use Linux

- Default operating system on virtually all HPC systems
- Extremely flexible and not overbearing
- Fast and powerful
- Many potent tools for software development
- You can get started with a few basic commands and build from there

Secure Shell (SSH)

- To a remote system, use Secure Shell (SSH)
- From Windows
  - Non-GUI SSH application: Windows PowerShell
  - GUI SSH application: PuTTY
  - Putty is preferred method.
    - Hostname: `login.rc.colorado.edu`

      or…
    - Hostname: `tlogin1.rc.colorado.edu`
- From Linux, Mac OS X terminal, ssh on the command line

RC Access: Logging in

- If you have an RMACC RC account already, login as follows from a terminal:

```
$ ssh <username>@login.rc.colorado.edu
# Where username is your identikey
```

- If you do not have an RMACC RC account use one of our temporary accounts:

```
$ ssh user<XXXX>@tlogin1.rc.colorado.edu
# Where user<XXXX> is your temporary username
```

Useful SSH Options

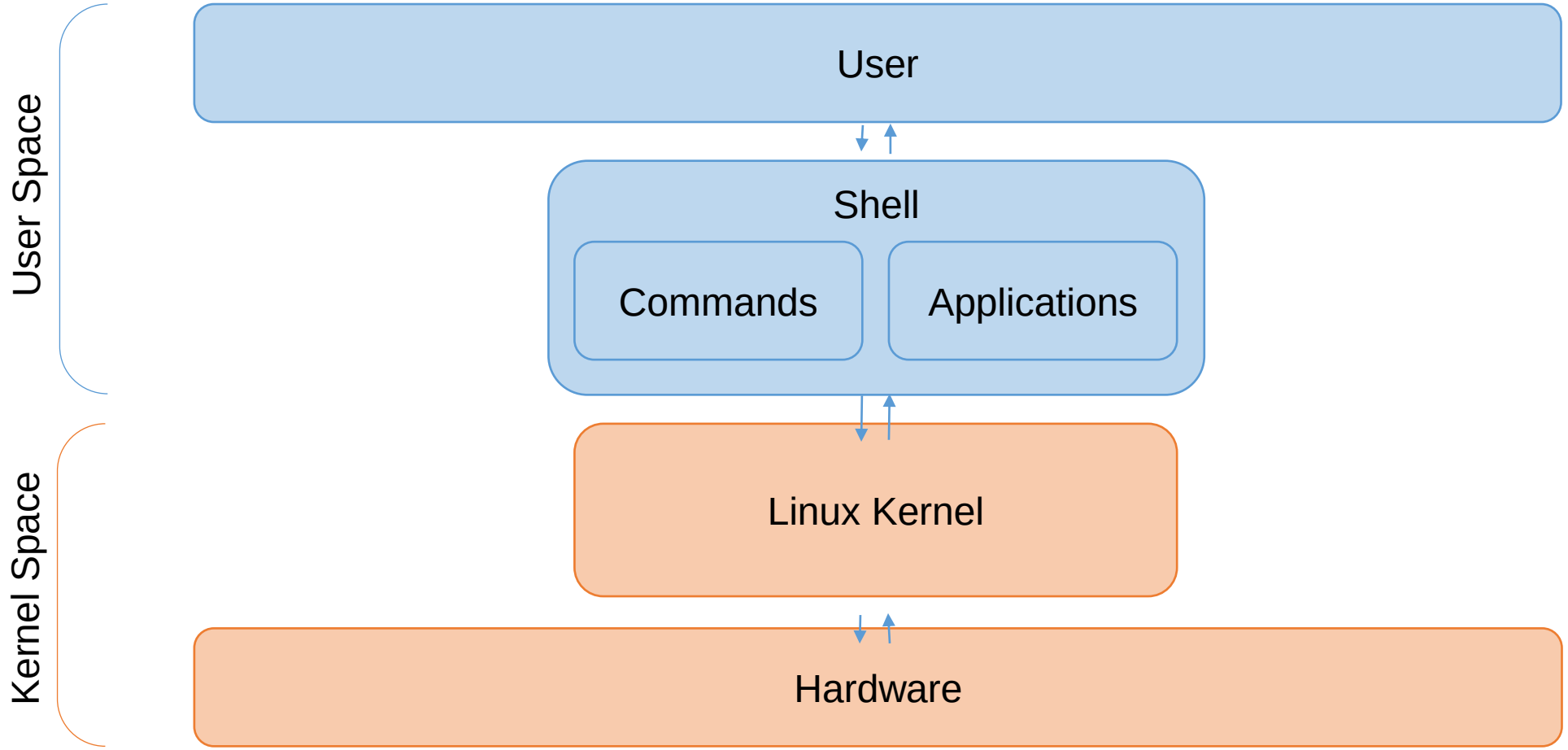
- `-X or -Y`
  - Allows X-windows to be forwarded back to your local display

- `-o TCPKeepAlive=yes`
  - Sends occasional communication to the SSH server even when you're not typing, so firewalls along the network path won't drop your "idle" connection

The Shell

- Parses and interprets typed input
- Passes results to the OS and returns results as appropriate.
- Shells
  - Bourne-Again (bash) – Widely used user friendly shell. Default on Summit.
  - T (tcsh) – C Shell with extended features and C syntax. Also very common.
- Features
  - Tab completion
  - History and command-line editing
  - Scripting and programming
  - Built-in utilities

Shells


User Space
Kernel Space
User
Shell
Commands
Applications
Linux Kernel
Hardware

Command Anatomy

- Case-sensitive
- Order of flags may be important
- Flags may not mean the same thing when used with different commands

The most important Linux command:

man

```
$ man <command>
$ man -k <keyword>
```

Note: You can google commands too!

https://man7.org/linux/man-pages/man1/man.1.html

Filesystem Commands

Command	Description
`pwd`	prints full path to current directory
`cd`	changes directory; can use full or relative path as target
`mkdir`	creates a subdirectory in the current directory
`rmdir`	removes an empty directory
`rm`	removes a file (`rm -r` removes a directory and all its contents)
`cp`	copies a file
`mv`	moves (or renames) a file or directory
`ls`	lists the contents of a directory (`ls -l` gives detailed listing)
`chmod/chown`	change permissions or ownership
`df`	displays filesystems and their sizes
`du`	shows disk usage (`du -sk` shows size of a directory and its contents in KB)

File Editing Commands

Command	Description
`less`	displays a file one screen at a time
`cat`	prints entire file to the screen
`head`	prints the first few lines of a file
`tail`	prints the last few lines of a file (with -f shows in realtime the end of a file that may be changing)
`diff`	shows differences between two files
`grep`	prints lines containing a string or other regular expression
`tee`	prints the output of a command and copies the output to a file
`sort`	sorts lines in a file
`find`	searches for files that meet specified criteria
`wc`	count words, lines, or characters in a file

Environments

- Set up using shell and environment variables
  - shell: only effective in the current shell itself
  - environment: carry forward to subsequent commands or shells
- Set default values at login time using `.bash_profile` (or `.profile`). Non-login interactive shells will read `.bashrc` instead.
- `var_name[=value]` (shell)
- `export VAR_NAME[=value]` (environment)
- `env` (shows current variables)
- `$VAR_NAME` (refers to value of variable)

Important variables

- `PATH`: directories to search for commands
- `HOME`: home directory
- `DISPLAY`: screen where graphical output will appear
- `MANPATH`: directories to search for manual pages
- `LANG`: current language encoding
- `PWD`: current working directory
- `USER`: username
- `LD_LIBRARY_PATH`: directories to search for shared objects (dynamically-loaded libs)
- `LM_LICENSE_FILE`: files to search for FlexLM software licenses

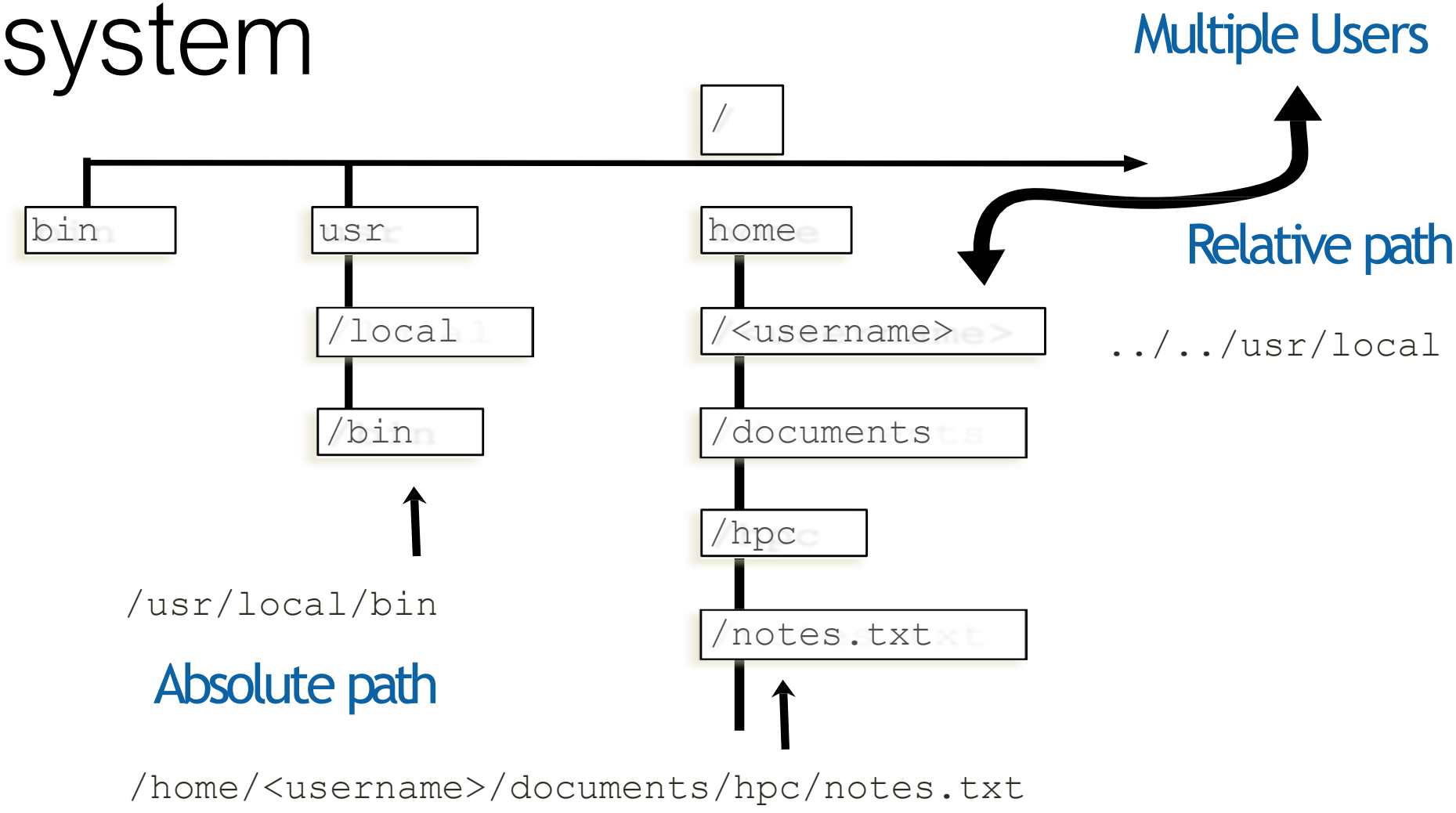
The Linux Filesystem

- System of arranging files on disk
- Consists of directories (folders) that can contain files or other directories
- Levels in full paths separated by *forward* slashes, e.g.

  /home/nunez/scripts/analyze_data.sh
- Case-sensitive; spaces in names discouraged
- Some shorthand:

Symbol	Description
.	Current directory
..	The directory 1 Level Above
~	The home directory
-	Previous directory when used with `cd`

Filesystem


Multiple Users
/
bin
usr
home
/local
/<username>
/bin
/documents
/hpc
/notes.txt
Relative path
../../usr/local
/usr/local/bin
Absolute path
/home/<username>/documents/hpc/notes.txt

Navigating the Filesystem

- Examples:
  - ls
  - mkdir
  - cd
  - rm
- Permissions (modes)

File Editing

- **nano** – simple and intuitive to get started with; not very feature-ful; keyboard driven
- **vi/vim** – universal; keyboard driven; powerful but some learning curve required
- **emacs** – keyboard or GUI versions; helpful extensions for programmers; well-documented
- **LibreOffice** – for WYSIWYG
- Use a local editor via an SFTP program to remotely edit files.

Modes/Permissions

- 3 classes of users:
  - User (u) *aka “owner”*
  - Group (g)
  - Other (o)
- 3 types of permissions:
  - Read (r)
  - Write (w)
  - Execute (x)

Modes

- `chmod` changes modes:

To add write and execute permission for your group:

```
chmod g+wx filename
```

To remove execute permission for others:

```
chmod o-x filename
```

To set only read and execute for your group and others:

```
chmod go=rx filename
```

Part 2: Job submission

RMACC Summit Supercomputer

- 450 compute nodes (Intel Xeon Haswell)
- 24 cores per node
- 11,400 total cores
- Omni-Path network
- 1.2 PB scratch storage
- GPFS File system

- 67% CU, 23% CSU, 10% RMACC

Additional Types of RMACC Summit Compute Nodes

- 10 Graphics Processing Unit (GPU) Nodes
  - NVIDIA Tesla K80 (2/node)
- 5 High Memory Nodes
  - 2 TB of memory/node, 48 cores/node
- 20 Phi Nodes
  - Intel Xeon Phi
  - 68 cores/node, 4x threads/core

Working on RC Resources

- When you first log in, you will be on a login node. Your prompt will look like this (e.g.):

```
[user0049@tlogin1 ~]$
```

- The login nodes are lightweight virtual machines primarily intended to serve as ‘gateways’ to RC resources. If you plan to work on Summit (most will), your first step should always be to move to a Summit ‘scompile node’:

```
[user0049@tlogin1 ~]$ ssh scompile
```

- Now go to your working directory and download the material for this workshop:

```
[user0049@shas0137 ~]$ cd /scratch/summit/$USER
[user0049@shas0137 ~]$ git clone
https://github.com/ResearchComputing/RMACC2021_Intro_to_HPC
[user0049@shas0137 ~]$ cd /RMACC2021_Intro_to_HPC/job_submission
```

Useful Slurm Commands: `sbatch`

- `sbatch`: submit a batch job to slurm
- Submit your first job! :

```
$ sbatch submit_hostname.sh
```

- Script contains most of the parameters needed to define a job
- Usually do not need additional flags.

http://slurm.schedmd.com/sbatch.html

Anatomy of a job script (submit_hostname.sh)

```
#!/bin/bash
#SBATCH --ntasks=1                          # Number of requested tasks
#SBATCH --time=0:01:00                      # Max wall time
#SBATCH --partition=shas-testing            # Specify Summit Haswell nodes
#SBATCH --output=test_%j.out                # Rename standard output file

# Updated by:   Shelley Knuth, 17 May 2019
# Purpose:  To demonstrate how to run a batch job on RC resources

# Purge all existing modules
module purge

# Run commands
echo "This is a test of user $USER"
```

SBATCH Options

Specified at command line or in job script as…
`#SBATCH <options>` …where options include:

- Allocation: `--account=<account_name>`
- Partition: `--partition=<partition_name>`
- Sending emails: `--mail-type=<type>`
- Email address: `--mail-user=<user>`
- Number of nodes: `--nodes=<nodes>`
- Number of Tasks `--ntasks=<number-of-tasks>`
- Quality of service: `--qos=<qos>`
- Reservation: `--reservation=<name>`
- Wall time: `--time=<wall time>`
- Job Name: `--job-name=<jobname>`

More on slurm commands:
https://slurm.schedmd.com/quickstart.html

FYI: You do NOT actually type <> above – this designates something specific you as a user must enter about your job

Available Partitions (--partition)

Partition	Description	# of nodes	cores/node	GPUs/node
shas	General Compute (Haswell)	~450	24	0
sgpu	GPU-enabled nodes	10	24	effectively 4
smem	High-memory nodes	5	48	0
sknl	Phi (Knights Landing) nodes	20	68	0

Sub-Partitions

Partition	Description	Max wall time	Max jobs/user	Max nodes/user
shas-testing sgpu-testing sknl-testing	For quick turnaround when testing	30 M	1	2 12 cores/node
shas-interactive	For interactive jobs (command or GUI)	4 H	1	1 core

Software

- Software is usually installed locally by Users in their /projects/$USER directory
- Generic versions of some software available with Lmod module software
- Lmod Commands:

Command	Description
module avail	Lists available software with current compilers and MPIs
module load <software-name>	Load software module into environment
module remove <software-name>	Removes software module from environment
module purge	Removes all software modules loaded
module spider <software-name>	Searched entire software tree for software

Practice Job Submission Examples

Write your first job script!

- Create a Slurm job script and submit it as a job, with the following instructions:

1. Name it 'submit_sleep.sh'

2. The job should contain the following commands:

```
echo "Running on host" `hostname`
echo "Starting Sleep"
sleep 30
echo "Ending Sleep. Exiting Job!"
```

Details on job script parameters are in the next slide

Details of submit_sleep.sh

1. The job will run on 1 core of 1 node
2. We will request a 1 minute wall time
3. Run on the shas-testing partition
4. Set the output file to be named “sleep_ID.out”
5. Name your job “sleep”
6. Bonus: Email yourself when the job ends
7. Contains the following commands →
8. Submit using the ‘rmacc’ reservation *(This is only for this workshop):*

```
echo "Running on host" `hostname`
echo "Starting Sleep"
sleep 30
echo "Ending Sleep. Exiting Job!"
```

```
$ sbatch --reservation=rmacc submit_sleep.sh
```

Solution can be found in “./solutions” subdirectory

Running and monitoring jobs

- Submit the job:

```
$ sbatch submit_hostname.sh
```

- Check the status of the job:

```
$ squeue / $ squeue -u <user> / $ squeue -q <qos> # or
$ sacct / $ sacct -u <user> # or
$ scontrol show job <job number>
```

- Look at the job output:

```
$ more hostname_xxxxxx.out # where xxxxxx is your Job Id
```

More on slurm commands: https://slurm.schedmd.com/quickstart.html

Running an external program

- Let's run R on an R script
- This script calls and runs the script *R_program.R*
- Let's examine the batch script submit_R.sh
  - Note how R is loaded
- Go ahead and submit the batch script submit_R.sh

Running an mpi job

- For cases where you have a code that is parallelized, meaning it can run across multiple cores.
- Number of tasks always > 1. E.g.,

```
#SBATCH --ntasks=4
```

- Will always need to load a compiler and mpi. E.g.,

```
module load intel impi
```

- Executable preceded with mpirun, srun, or mpiexec. E.g.,

```
mpirun –np 4 python yourscript.py
```

- Examine and run the example ‘submit_python_mpi.sh’

```
$ sbatch --reservation=rmacc submit_python_mpi.sh
```

Interactive jobs

- Sometimes we want our job to run in the background
- Sometimes we want to work on program in real time
    - Great for testing, debugging
- For example, let's run the R job we previously ran as a batch job, but this time let's do it interactively…

Running an interactive job

- To work with R interactively, we request time from Summit
- When the resources become available the job starts
- Commands to run:

```
$ sinteractive –-time=00:10:00 --reservation=rmacc
```

- Once we receive a prompt, then:

```
$ module load R
$ cd ./progs
$ Rscript R_program.R
```

- Once we finish we must exit! (job will time out eventually)

```
$ exit
```

Tools for submitting many small jobs at once

- GNU Parallel
  - https://curc.readthedocs.io/en/latest/software/GNUParallel.html
- CURC Load Balancer
  - https://curc.readthedocs.io/en/latest/software/loadbalancer.html
- Slurm job arrays
  - https://slurm.schedmd.com/job_array.html

Thank you!

- Please fill out the survey: http://tinyurl.com/curc-survey18
- Contact information: rc-help@Colorado.edu
- Slides and Examples from this course: https://github.com/ResearchComputing/RMACC2021_Intro_to_HPC

- Slurm Commands: https://slurm.schedmd.com/quickstart.html