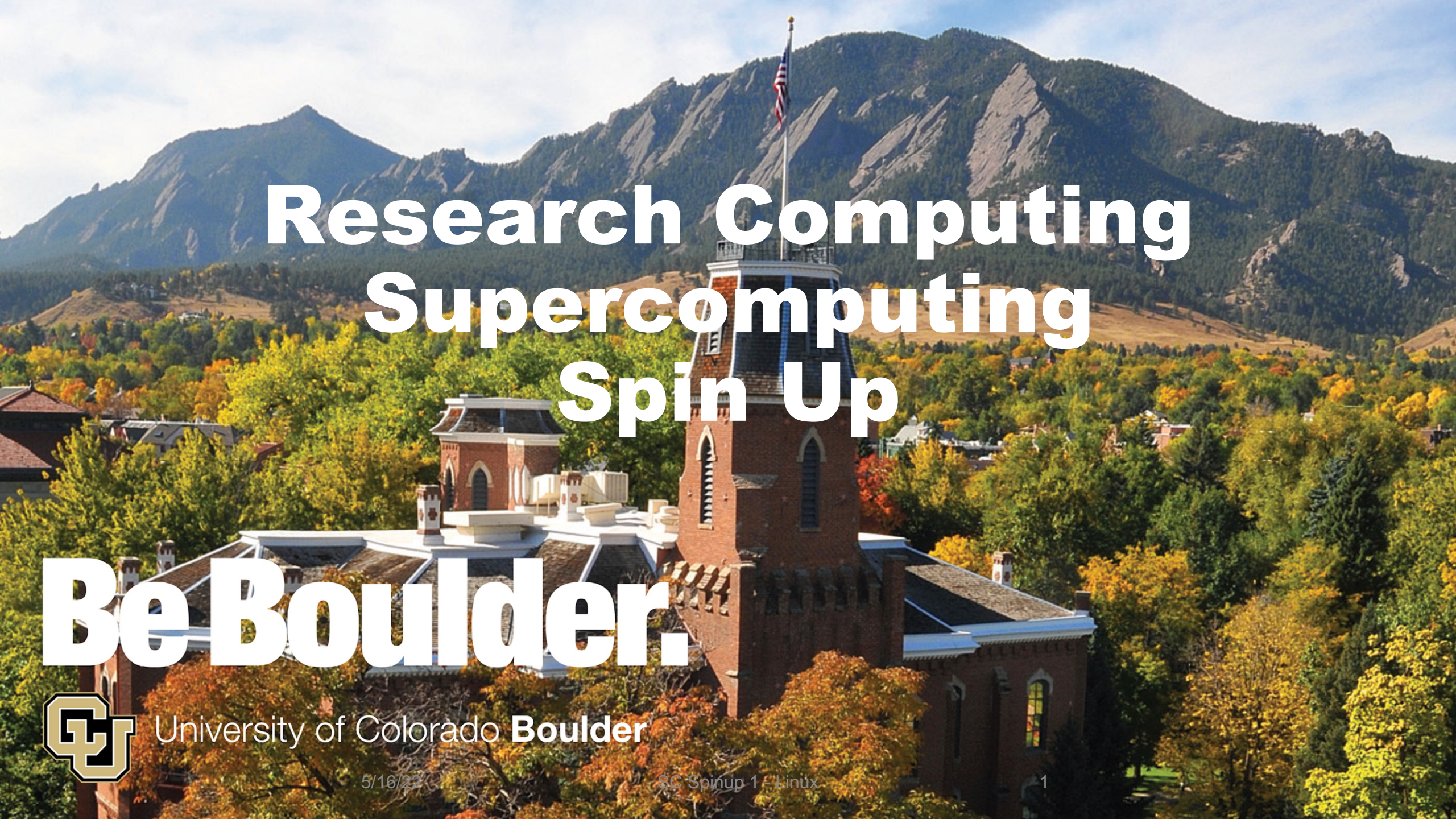

Research Computing
Supercomputing
Spin Up
Be Boulder.
University of Colorado Boulder

Supercomputing Spinup Part I: Working with Linux

Layla Freeborn
Contact: rc-help@colorado.edu
Website: https://www.colorado.edu/rc

Slides and other files available for download and viewing:
https://github.com/ResearchComputing/Supercomputing_Spinup

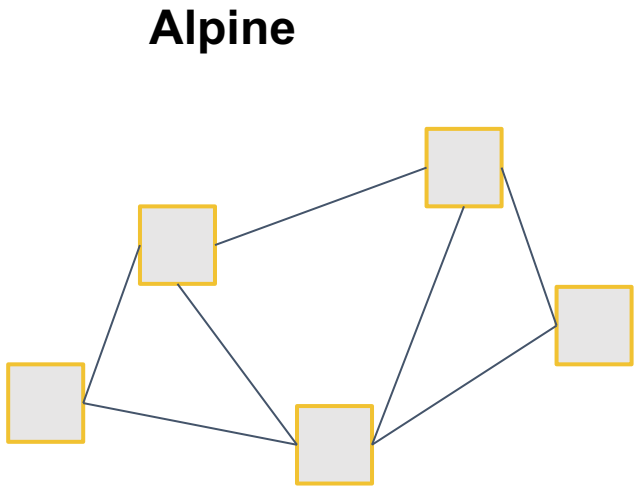
Working with Linux Outline

- Intro to Research Computing
- Opening a Terminal
- Why use Linux
- Basic Linux commands
- File editing
- Linux filesystems
- Environment variables
- Modes (aka Permissions)
- Intro to the shell and shell scripts
- File editing
- Bash scripting basics

What is Research Computing?

- Provide services for researchers that include:
  - Large scale computing
  - Data storage
  - High speed data transfer
  - Data management support
  - Consulting
  - Training
- Our compute clusters include Alpine, Summit, and Blanca ”condo” cluster
- Archive and active storage provided by PetaLibrary

HPC Cluster: Alpine



- Alpine is the 3rd-generation HPC cluster at CURC, following:
  - Janus
  - RMACC Summit
- Alpine is a heterogeneous cluster with hardware currently provided by CU Boulder
- Access currently limited to CU Boulder users
- Additional contributions provided from Colorado State University and Anschutz Medical Campus are planned for the near future

RMACC Cyber Infrastructure

- https://ask.cyberinfrastructure.org/c/rmacc/65
- This forum provides opportunity for RMACC members to converse amongst themselves and with the larger, global research computing community.
- The “go to” general Q&A platform for the global research computing community - researchers, facilitators, research software engineers, CI engineers, sys admins and others.

Opening a Terminal

- Mac: Go to Applications → Utilities → Terminal

- Windows: Download a terminal emulator
  - PuTTY: https://www.putty.org
  - Git BASH: https:gitforwindows.org

- For practice, you can use an online emulator such as https://cocalc.com/app?anonymous=terminal

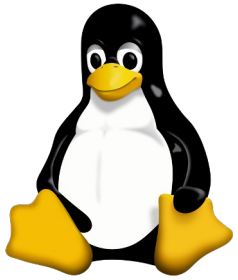
Logging into CURC

- ssh <identikey>@login.rc.colorado.edu
- Enter your identikey password
- Authenticate by accepting the Duo push to your smartphone
  - Can also authenticate by text message, phone call, or token
- More info here: https://curc.readthedocs.io/en/latest/access/logging-in.html

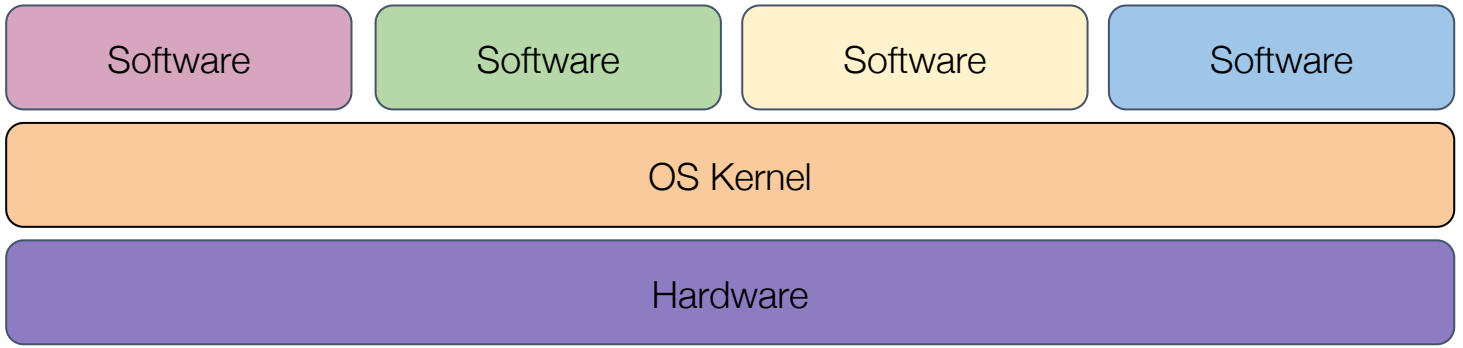
What happens when you log in?

- Login is authenticated (password or key)
- Assigned to a tty
- Shell starts
- Environment is set up
- “Message of the Day” prints
- Prompt

What is Linux?



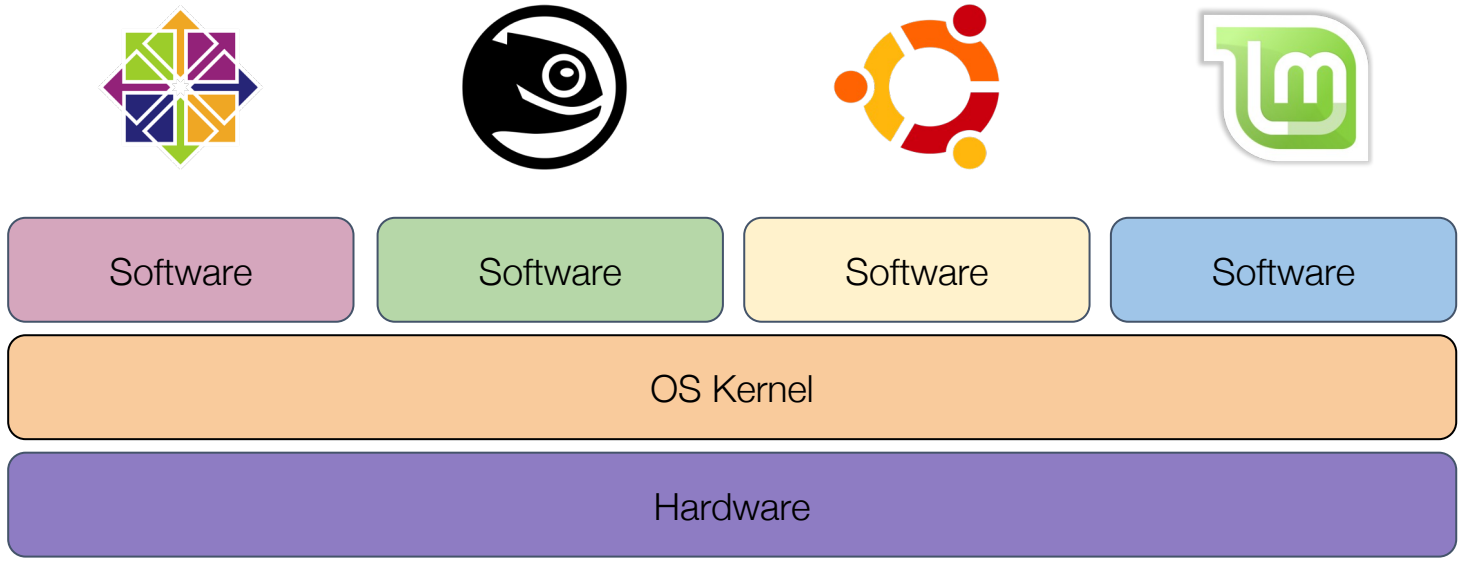
- Part of the Unix-like family of operating systems.
- Started in early ‘90s by Linus Torvalds.
- Typically refers only to the kernel with software from the GNU project and elsewhere layered on top to form a complete OS. Most is open source.



images courtesy of wikicommons

What is Linux?

- 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.



images courtesy of wikicommons

Why Use Linux?

- Default operating system on virtually all HPC systems and the foundation for many business services globally
- Extremely flexible
- Fast and powerful
- Many tools for software development
- You can get started with a few basic commands and build from there

Anatomy of a Linux command

- command [flags] [target(s)]

  ls –l myworkdir/

- Case is important!
- Help on commands is available through the “man” command (short for manual)

  man ls

The Linux Filesystem

- System of arranging files on disk
- Consists of directories (folders) that can contain files or directories
- Levels in full paths separated by forward slashes:

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

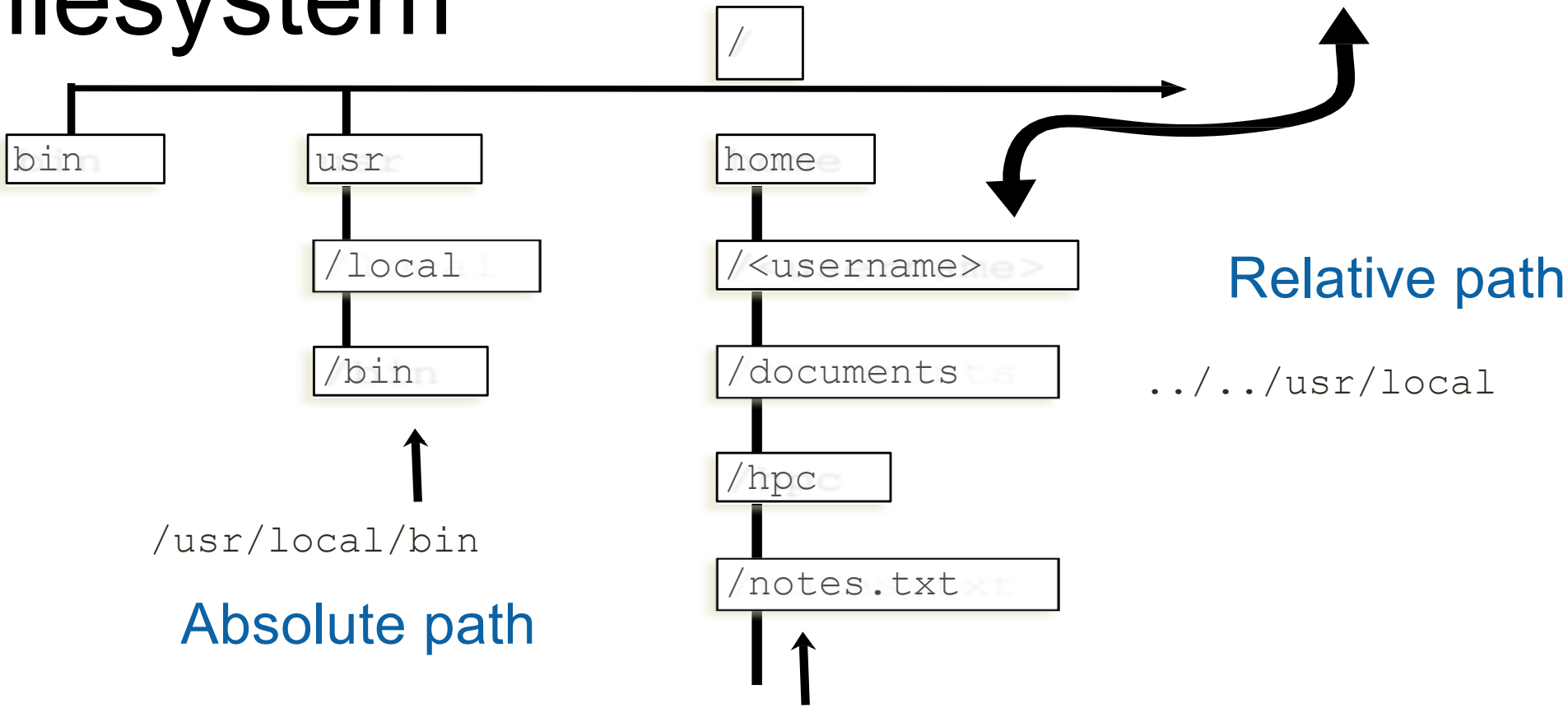
  `.` (the current directory)

  `..` (the directory one level above)

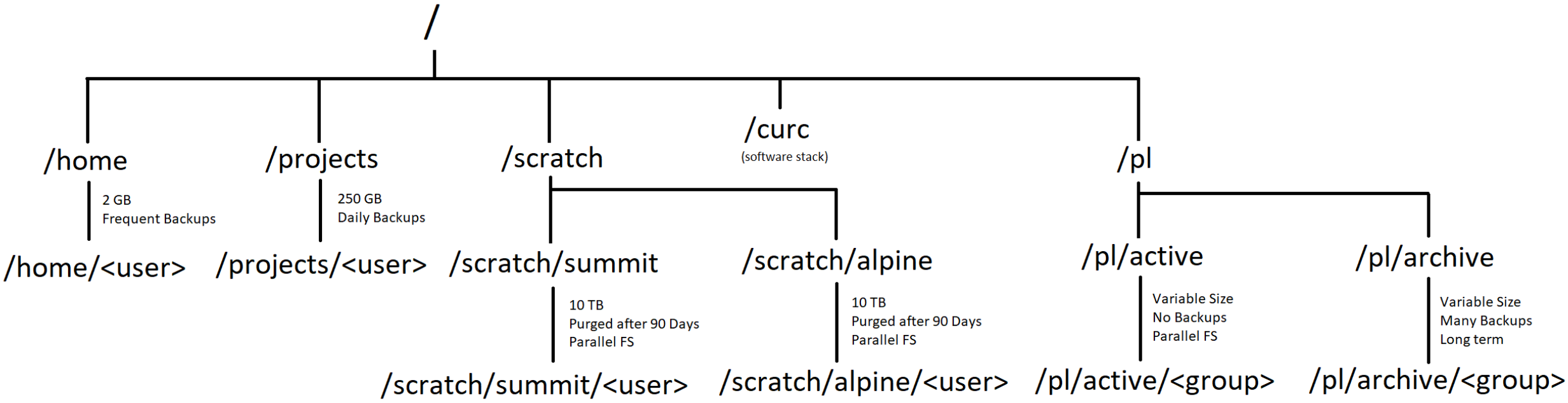
  `~` (home directory)

  `-` (previous directory, when used with `cd`)

Filesystem


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

Your personal directories on CURC



Environment variables

- Environment variables store important information needed by Linux users and programs
- Type env to see your currently set up environment variables
- Useful environment variables:

PATH	directories to search for commands
HOME	home directory
PWD	current working directory
USER	username
LD_LIBRARY_PATH	directories to search for dynamically-loaded libraries

File and Directory related commands

- **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 of 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)

File-viewing commands

- **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

  ps –df | grep xx
- **sort** – sorts lines in a file
- **find** – searches for files that meet specified criteria
- **wc** – count words, lines, or characters in a file

Process and Program related commands

ps – lists processes (ps -ef lists all running processes)

top – shows processes currently using the CPU

kill – sends a signal to a process (kills process by default). Target is Process-ID; found in 2nd column of ps -ef output.

time – shows how much wall time and CPU time a process has used

Exercise 1: Navigation

1. Change to your projects directory *without* typing your user name out
2. Print the path to your current directory
3. List the contents of this directory
4. Change to your home directory and create a new directory (you can pick the name). How can you be sure the new directory is there?
5. Move the directory to your project space
6. Remove the directory you just created. *Be careful!*

Access the example scripts

- How to get there: github.com/ResearchComputing/Supercomputing_Spinup
- Navigate to your workspace and go into the scripts directory for our demos
- Git clone the repository:

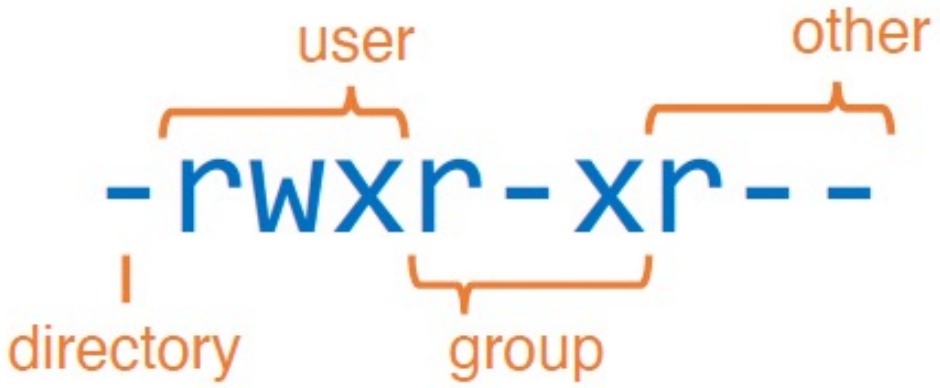
```
git clone https://github.com/ResearchComputing/Supercomputing_Spinup.git
```

Exercise 2: File Viewing

1. Change to the “bash_spinup/scripts” directory
2. Print out the entire “test.sh” file
3. Print out the last 3 lines of “local_vs_global.sh” file
4. Find how many words are in “case_example.sh”

Modes (aka permissions)

- View file/directory permissions `ls -l`
- 3 classes of users:
  - User (u) aka “owner”
  - Group (g)
  - Other (o)
- 3 types of permissions:
  - Read (r)
  - Write (w)
  - Execute (x)



Modes (continued)

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
```

Intro to Shells and Shell Scripts

A **shell** is the environment in which commands are interpreted in Linux.

GNU/Linux provides numerous shells; the most common is the Bourne Again shell (bash).

Other common shells available on Linux systems include:

- sh, csh, tcsh, ksh, zsh

Shell scripts are files containing collections of commands for Linux systems that can be executed as programs.

Shell scripts are powerful tools for performing many types of tasks.

- Can be programmed interactively, directly on the terminal.
- It can also be programmed by script files. The first line of the file must contain `#!/bin/bash`
- The program loader recognizes the `#!` and will interpret the rest of the line (`/bin/bash`) as the interpreter program.
- If a line starts with `#`, it is a comment and is not run.

```
#!/bin/bash                    Shell to run
# the files in /tmp.           Comments
cd /tmp                        Change directories
ls                             List everything in /tmp
```

Alternatives for Scripting

- `csh/tcsh` — C-shell (tcsh: updated version of csh)
- `ksh` — Korn shell; related to sh/bash
- `perl` — exceptional text manipulation and parsing
- `python` — excellent for scientific and numerical work
- `ruby` — general scripting
- `make` — building executables from source code

Exercise 3: Permissions and Running Bash Scripts

1. Change directory to `~/<repo for this class>`
2. Use `less` to view the contents of `hello_world.txt`
3. Use `cat` to show the contents of `hello.sh` in `bash_spinup/scripts`
4. Try to run `hello.sh` by typing `./hello.sh` at the command line
5. Add execute permission to `hello.sh` using `chmod`
6. Try to run `hello.sh`

File editing

- **nano** – simple and intuitive to get started with; not very feature-rich; keyboard driven
- **vi/vim** – universal; keyboard-driven; powerful but has a learning curve
- **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

File Editing with Nano

`test.sh`

- type `nano <filename>` at the prompt.
- You can edit text as you would in, e.g. MS Word.
- When you are finished, type ctrl-o to write, ctrl-x to exit. See commands at the bottom of the screen.
- How can we run the script?

Exercise 4: File Editing with Nano

1. Edit the contents of `hello_world.txt` contents with `nano` (you can edit it to say anything!)
2. Run the program "`hello.sh`" by typing `bash hello.sh` or `./hello.sh` at the command line

Bash Scripting Basics: Local vs. Global Variables

- A variable can contain a number, a character, a string of characters.
- Environment variables are global- carry forward to subsequent commands or shells
- Shell variables are local- only effective in the current shell itself
- A variable declared as local is one that is visible only within the block of code in which it appears. It has local "scope". In a function, a local variable has meaning only within that function block.

Exercise 5: Local Variables Scope

`variables_scope.sh`

1. Run it (make sure it's executable)
2. Take a look at the file
3. Why did `var2` change but `var1` stayed the same?
4. Edit the script so that `var2` stays the same *without editing* `my_function`

Bash Scripting Basics: Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value by enclosing the command with: (( ))

```
$ sqr_two=$(( 2 * 2 ))
$ echo ${sqr_two}
$ 4
```

Note that Bash only does integer math by default, however it is easy to do floating point math with the Bash calculator tool (called `bc`)

```
$ echo "5.6/9.4" | bc -l
$ .59574468085106382978
```

Thank you!

Layla Freeborn
https://www.colorado.edu/rc

Survey:
http://tinyurl.com/curc-survey18

Additional Bash learning resources:
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html (general)
https://www.shell-tips.com/2010/06/14/performing-math-calculation-in-bash/ *(math)*

Bash kernel for jupyter notebooks *(install anaconda first)*:
https://github.com/takluyver/bash_kernel