

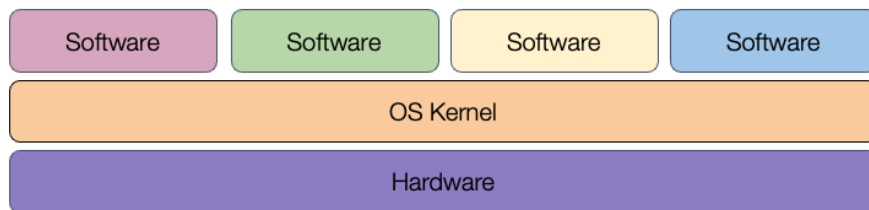
Module 3: Working with Linux

Learning Goals

- Why Linux?
- Working with Files
- Working with Scripts

What is Linux?

- Created by Linus Torvalds (1991)
- “Unix”-based operating system (like Mac OS)
- Supports a variety of hardware and software systems



The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.

Linux Distros

- Variety of distributions, or distros, available
- Embedded systems (Raspberry PI)
- “Windows” replacement (Ubuntu)
- Commercial/Industry Supported (RedHat)



images courtesy of wikicommons

Red Hat Enterprise Linux
Software- und System-Entwicklung (Software and Systems Development)

Why Use Linux?

- Most common Operating System for HPC systems
- Extremely flexible, fast, and powerful
- Built-in support for many software development workflows



images courtesy of wikicommons

Add that Linux is open source

Opening a Terminal

- Mac: Go to **Applications** → **Utilities** → **Terminal**
- Windows: Download a terminal emulator (or use Powershell)
 - PuTTY: <https://www.putty.org>
 - Git BASH: <https://gitforwindows.org>
 - Open OnDemand: <https://ondemand.rc.colorado.edu/>
 - RMACC: <https://ondemand-rmacc.rc.colorado.edu/>

<https://curc.readthedocs.io/en/latest/access/logging-in.html>

Logging into CURC

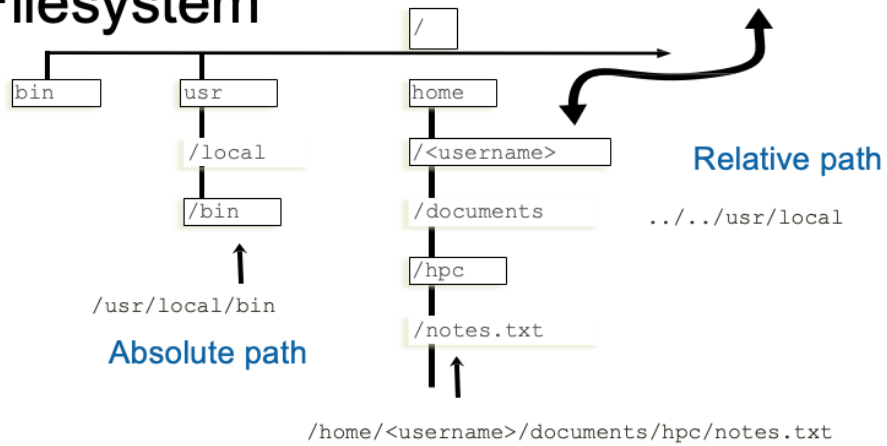
- `ssh <rc_username>@login.rc.colorado.edu`
- Enter your password
- Authenticate by accepting the Duo push to your smartphone

<https://curc.readthedocs.io/en/latest/access/logging-in.html>

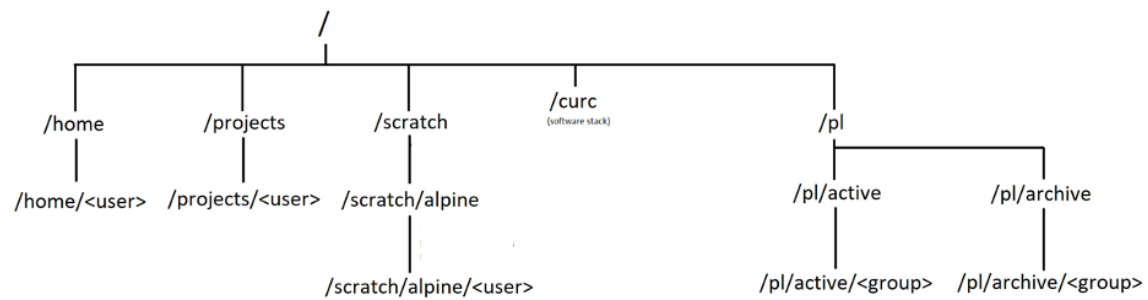
The Linux Filesystem

- System of arranging files and directories (folders)
- 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



Your personal directories on CURC



Home: Good for 'can't lose' files

projects: good for storing scripts, self-installed software, some data

scratch- no backed up! highly performant- good for jobs with lots of I/O

Anatomy of a Linux command

- command [flags] [target(s)]

```
ls -l myworkdir/
```

- Case is important!
- Use “man” command to view a command’s manual

```
man ls
```

-l : Displays the mode, number of links, owner, group, size (in bytes), and time of last modification for each file

-L: Lists the file or directory contents that the link references. This is the default action.

Change Directories

```
cd <path/to/take>
```

```
cd /projects/$USER
```

Make Directories

```
mkdir <path/directory_name>
```

```
mkdir rc_temp
```

Cloning a git repository

- How to get there: github.com/ResearchComputing/Supercomputing_Spinup

- Clone the repository:

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

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

List Files and Directories

ls [option] [file/directory]

-l (long format) -a (all files)
-h (readable file size) -r (reverse sort)
-S (sort by file size)

Copy Files

```
cp <source> <destination>
```

```
cp README.md ../
```

```
cp README.md ../Docs.md
```


Remove Files

```
rm <path/to/file>
```

```
rm Docs.md
```

```
rm -r <directory>
```

Be careful when
using a recursive (-r)
delete. It can delete
everything!

Text Editors

- **nano** – Beginner friendly
- **vi/vim** – Powerful, but steep learning curve
- **emacs** – Extendable, tons of additional features
- **VS Code** – via OnDemand
- Use local text editor and copy files manually to Alpine

Create a Text File

vim notes.txt

nano notes.txt

Head Command – First X Lines of File

```
head <path/to/file>
```

```
head notes.txt
```

```
head -n 3 notes.txt
```

Tail Command – Last X Lines of File

```
tail <path/to/file>
```

```
tail notes.txt
```

```
tail -n 3 notes.txt
```

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.

If you have a newer mac and you're working in the terminal, you might notice it'll show you a message about zsh being the default shell

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

- Executed interactively (terminal) or programmatically (scripts)
- In shell scripts, the first line 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.

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

The `hello.sh` file you `chmod`'d was a bash shell script.

This implies that the Bash script is running in a separate independent shell. This separate shell terminates at the end of the script, leaving the parent shell, the shell we're in, unaffected.

```
pwd
echo $$
PID OF CURRENT PROCESS
```

Alternatives for Scripting

- `cs`/`tcsh` C-shell (`tcsh`: updated version of `cs`)
- `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

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)

`-rwxr-xr--`

user other

directory group

d Means directory, - means file

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
```

Run test.sh

```
chmod u+x test.sh
```

```
./test.sh
```

Run hello.sh

```
chmod u+x hello.sh
```

```
./hello.sh
```

Local vs Global Variables

- A variable can contain a number, a character, a string of characters.
- Environment variables are global – effective in subsequent shells
- Shell variables are local- only effective in the current shell itself

I gave you the example of \$USER being an environmental variable- it's a variable stored in the system that's there every time you login, no matter whether you're on a compute node or the login node.

Variables that you set in your shell or shell script, on the other hand, are examples of local variables. Local variables are those whose scope is within the function where it is declared(can be accessed within the declared block or function in the program)

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:

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

have them do `echo $USER`

Run variables_scope.sh

```
chmod u+x variables_scope.sh
```

```
./variables_scope.sh
```