



Linux basics for research computing

Ryan Bradley

Spring 2023

Objectives

In this seminar,
we will learn how to talk to machines.

- 1 Linux is a practical tool that makes it easy to automate your computations.
- 2 Linux and its associated ecosystem of tools is the *de facto* standard for computation.
- 3 Linux will help you think systematically about research computation.

What is Linux?

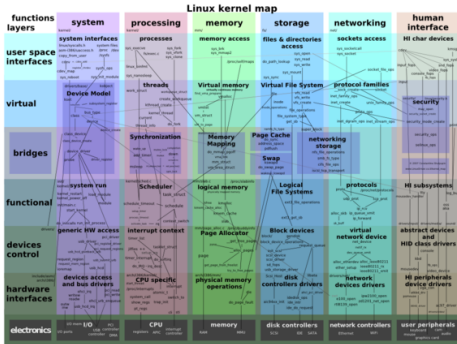


Figure 1: Map of the kernel, via Wikipedia.

- A monolithic kernel which controls the hardware, arbitrates conflicts, and schedules common resources.
- Always present in memory.
- Most popular operating system on supercomputers.

Linux is a standard

- There are many varieties called distributions.
- Distributions package the kernel with a package management system and configuration tools.
- Popular distributions
 - RedHat, Fedora, CentOS,
 - Debian, Ubuntu, Mint
 - Slackware, OpenSuSE, SLES

Linux is a free, open source

- Linux is Free and Open Source Software (FOSS)
- Released under the GNU General Public License
- Free as in beer? or liberty? or markets?
- Linux is collectively developed, customized, and often sold in tandem with support.
- FOSS mirrors the scientific world, in which collaboration and scientific progress depend on cooperation at a massive scale.
- Copyright and licenses answer the twin questions:
 - ① Can you distribute this code in a closed or proprietary software?
 - ② How do you credit the author of the code?

Package managers

- Software that automates the installation and configuration of other software.
- Package managers orchestrate the cooperation of libraries and support across many packages so that you *don't repeat yourself* (DRY).
- The interfaces between different software facilitates a modularity that makes it possible to leverage many pieces of technology to build new software.
- Hence software development and research exists on the expanding boundary of a very large metaphorical, cooperative text.
- There are two standards: **deb** (Ubuntu) and **rpm** (centOS).
- Package managers and distributions can adapt Linux to special use-cases (e.g. Android).
- Linux has desktop and graphical environments: KDE, GNOME, XFCE, etc, and has equivalents for most graphical applications you might run on a personal computer.

Shells



Figure 2: A terminal, via Wikipedia.

All commands are little programs

- The command line interface (CLI) is the medium by which we talk to the machine.
- A shell is an *implementation* of a CLI.
- The shell exposes operating system services to a human or other programs.
- Emulates a terminal, which sent pre-processed instructions to a larger computer.
- The shell rises to the level of a programming language.
- The shell has variables and an internal state.

Commands

Commands are directives to perform tasks.

- The command prompt is a conversation with the shell, which repeats your commands to the kernel.
- There are many shells with different features:
 - `sh` , `csch` , `ksh` , `bash` , `tcsh`
- Advice: do not get creative with your shells until you master Linux. Start with `BASH` and `sh` .
- Commands are programs.
- Programs live on the path, a list of locations in the filesystem set by an environment variable.

Syntax

Commands are delimited by spaces.

```
command --option --flag <VALUE> --another <VALUE2> <arg1> <arg2> ...
```

- Syntax follows standards, not rules.
- Cf. `docopt` and Python's `click` (the author's favorite).
- There are many competing standards for *all kinds of software*, see `POSIX` .

Next we will survey the most useful commands in order to provide a basis for automating our research with Linux.

Filesystem

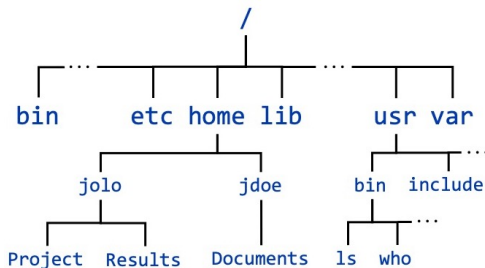


Figure 3: Example filesystem (via Cornell Virtual Workshop).

- File are arranged in an hierarchy, a tree with links (almost a directed graph).
- The top of the hierarchy is called “root” `/` (not to be confused with a superuser).
- Linux uses forward slashes to denote directories.
- Paths can be relative or absolute: `./` means “here”, `../` is the parent directory.
- Relative paths are *not unique*, which means that you can get to the same location in many ways.

Log on

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh lssh — 87x12
rpb222@ssh9:~$ ssh rpb222@sol.cc.lehigh.edu
rpb222@sol.cc.lehigh.edu's password:
Last login: Sun Feb  5 21:10:33 2023 from 128.180.6.12
System Architecture: avx2
Processor Family: haswell

Account total usage (SUs):
lts: 372672.07777/no limit

[rpb222@sol ~]$
```

Use `ssh` to log on to `sol` with this command:

```
ssh abc123@sol.cc.lehigh.edu
```

Alternately, you can use a MacOS desktop available in the lab.

Read the manual

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
-1 list one file per line. Avoid '\n' with -q or -b

--help display this help and exit

--version
    output version information and exit

The SIZE argument is an integer and optional unit (example: 10K is 10*1024).
Units are K,M,G,T,P,E,Z,Y (powers of 1024) or KB,MB,... (powers of 1000).

The TIME_STYLE argument can be full-iso, long-iso, iso, locale, or +FORMAT.
Manual page ls(1) line 190/235 82% (press h for help or q to quit)
```

Reveal the “manpages” with: `man ls`

- *This is the most important thing you will learn today!*
- Use the `q` button to exit.
- Use `/<search_term>` followed by `Enter` to search.
- *N.b* we use brackets for arguments, `/<search_term>`, and a question mark for optional arguments throughout the rest of this seminar: `... <path?> ...`

List files



```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile demo deploy JOB_TMPDIR lts_proj monitor ondemand R stage
[rpb222@sol ~]$
```

List files: `ls <path?>`

List files

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile demo deploy JOB_TMPDIR lts_proj monitor ondemand R stage
[rpb222@sol ~]$ ls -a
.          .bashrc    deploy     lts_proj   R           .wget-hsts
..         .cache    .ipython_checkpoints monitor     .Renvirom
.apptainer compile    .ipython   .nv        .spack
.bash_history .condarc  JOB_TMPDIR ondemand   .ssh
.bash_logout .config   .lmod.d    .pki       stage
.bash_profile demo      .local     .python_history .viminfo
[rpb222@sol ~]$
```

List hidden (“all”) files:

```
ls -a <path?>
```

List files

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
.bash_profile  demo      .local          .python_history  .viminfo
[rpb222@sol ~]$ ls -l
compile
demo
deploy
JOB_TMPDIR
lts_proj
monitor
ondemand
R
stage
[rpb222@sol ~]$
```

List files in a column: `ls -l <path?>`

List files

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls -all deploy/
total 20
drwxr-xr-x  4 rpb222 faculty 4096 Jan 25 12:44 .
drwx----- 22 rpb222 faculty 4096 Feb  6 07:19 ..
-rw-r--r--  1 rpb222 faculty   38 Jan 25 12:44 go.sh
drwxr-xr-x 10 rpb222 faculty 4096 Jan 25 11:19 spack
drwxr-xr-x  7 rpb222 faculty 4096 Jan 25 06:05 spackenv
[rpb222@sol ~]$
```

List files with metadata: `ls -al <path?>`

List files

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
lrwxrwxrwx 1 root    root      25 Jan 16 09:26 lts_proj -> /share/ceph/hawk/lts_proj
drwxr-xr-x 3 rpb222  faculty  4.0K Jan 23 11:11 ondemand
drwxr-xr-x 4 rpb222  faculty  4.0K Jan 25 12:44 deploy
drwxr-xr-x 3 rpb222  faculty  4.0K Feb  1 13:17 R
drwx----- 2 rpb222  faculty  4.0K Feb  2 06:30 JOB_TMPDIR
drwxr-xr-x 2 rpb222  faculty  4.0K Feb  5 21:09 demo
drwxr-xr-x 3 rpb222  faculty  4.0K Feb  5 21:11 stage
drwxr-xr-x 4 rpb222  faculty  4.0K Feb  6 06:55 compile
drwxr-xr-x 2 rpb222  faculty  4.0K Feb  6 06:56 monitor
drwxr-xr-x 2 rpb222  faculty  4.0K Feb  6 07:07 notes
-rw-r--r-- 1 rpb222  faculty    0 Feb  6 07:13 example-text.txt
[rpb222@sol ~]$
```

Sort the list by descending modification time:

```
ls -ltrh <path?>
```

- The author has limited memory, so we use a mnemonic (“later”).
- The human-readable flag (`-h`) also tells us the file size in an intuitive format.
- We can list multiple flags at once.



Echo some text

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ echo "Hello World"
Hello World
[rpb222@sol ~]$ █
```

Repeat some text: `echo "Hello, World!"`

- Essential for automation. When we write scripts, we use `echo` to repeat text to the “standard output” stream (`stdout`), the text that appears in our terminal.
- When a script runs asynchronously or autonomously, we need it to tell us what it’s doing.
- Similar to a `print` statement. Half of all debugging happens this way.



Present working directory

```

ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
deploy
JOB_TMPDIR
lts_proj
monitor
ondemand
R
stage
[rpb222@sol ~]$ pwd
/home/rpb222
[rpb222@sol ~]$ echo $PWD
/home/rpb222
[rpb222@sol ~]$ █
  
```

Echo your location in the filesystem: `pwd`

- Sometimes called “current working directory” in other contexts.
- We can also ask an “environment variable” to tell us this (more on this later).
- Usually included in the `$PS1`, or the prompt on the left.
- The `~` symbol means home (a.k.a. `$HOME`).

Change directory

```
ryanbradley — rpb222@sol:/home/rpb222/compile — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ pwd
/home/rpb222
[rpb222@sol ~]$ cd compile
[rpb222@sol compile]$ pwd
/home/rpb222/compile
[rpb222@sol compile]$ ls
rmc_pot_2020_1  rmc_pot_2020_1_source.tar.gz  test_run  test_run_RMC_POT_1.4.tar.gz
[rpb222@sol compile]$
```

Move to another directory: `cd <path?>`

- Without an argument, you return to “home”, typically `/home/<username>` .
- The argument can be an absolute path: `/home/<username>/some/folder`
- Or a relative path, starting with the dot operator (`.`): `./path/to/folder`
- Only relative paths are *path dependent* ... it matters where you are.

Make a directory

```
ryanbradley — rpb222@sol:/home/rpb222/compile/someprogram/v0.1/code — ssh rpb222@sol.cc...
[rpb222@sol ~]$ mkdir compile/someprogram/v0.1
mkdir: cannot create directory 'compile/someprogram/v0.1': No such file or directory
[rpb222@sol ~]$ mkdir -p compile/someprogram/v0.1
[rpb222@sol ~]$ cd compile/someprogram/v0.1
[rpb222@sol v0.1]$ ls
[rpb222@sol v0.1]$ mkdir code
[rpb222@sol v0.1]$ cd code
[rpb222@sol code]$ █
```

Make a directory: `mkdir <dirname>`

- We use the “prefix” flag (`-p`) to recursively generate nested directories.

Create a file

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile demo deploy JOB_TMPDIR lts_proj monitor notes ondemand R stage
[rpb222@sol ~]$ touch example-text.txt
[rpb222@sol ~]$ ls
compile deploy          JOB_TMPDIR  monitor    ondemand  stage
demo    example-text.txt  lts_proj   notes      R
```

Create and update the timestamp: `touch <filename>`

- This is mostly useful for demonstration purposes.
- This updates the modification time (more on timestamps later).

Copy a file

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile  deploy                JOB_TMPDIR  monitor    R
demo    example-text.txt     lts_proj   ondemand  stage
[rpb222@sol ~]$ mkdir notes
[rpb222@sol ~]$ cp example-text.txt notes/
[rpb222@sol ~]$ ls -l notes/
example-text.txt
[rpb222@sol ~]$
```

The copy commands duplicates a file:

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

- Overwrites the destination. **BEWARE!**
- Both arguments must be paths.
- If the second argument is a directory, the file retains its name.
- Use a trailing slash to be sure your directory name is correct.
- If available, tab (or auto-) completion can help.

Aside: copy and preserve timestamps

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile deploy          JOB_TMPDIR monitor  ondemand stage
demo    example-text.txt  lts_proj  notes    R
[rpb222@sol ~]$ cp -av example-text.txt notes/
'example-text.txt' -> 'notes/example-text.txt'
[rpb222@sol ~]$ ls -all notes/
total 12
drwxr-xr-x  2 rpb222 faculty 4096 Feb  6 07:07 .
drwx----- 22 rpb222 faculty 4096 Feb  6 07:07 ..
-rw-r--r--  1 rpb222 faculty 2312 Feb  2 12:40 example-text.txt
[rpb222@sol ~]$ █
```

Maintain metadata (“archive”):

`cp -av <source> <destination>`

- When you copy a file, it “creates” a duplicate with a timestamp for “now”.
- To preserve the metadata, use the “archive” feature, which preserves the creation and modification times, as well as other metadata (owner, permissions, etc).
- Flags are explained in the manpages.

Rename a file

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
deploy
JOB_TMPDIR
lts_proj
monitor
ondemand
R
stage
[rpb222@sol ~]$ pwd
/home/rpb222
[rpb222@sol ~]$ echo $PWD
/home/rpb222
[rpb222@sol ~]$
```

Move or rename a file with: `mv <source> <destination>`

- Overwrites the destination. **BEWARE!**
- You could alternately use many arguments to move **N** files to a single folder.

Remove

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile  deploy          JOB_TMPDIR  monitor  ondemand  stage
demo    example-text.txt lts_proj    notes    R
[rpb222@sol ~]$ rm example-text.txt
[rpb222@sol ~]$ ls
compile  demo  deploy  JOB_TMPDIR  lts_proj  monitor  notes  ondemand  R  stage
[rpb222@sol ~]$
```

Remove a file: `rm <path>`

- Obviously dangerous. You cannot get the file back. **BEWARE!**
- Use the “interactive” flag (`-i`) to prompt you before every file.

Remove

```
ryanbradley — rpb222@sol:/home/rpb222 — ssh rpb222@sol.cc.lehigh.edu — 87x12
[rpb222@sol ~]$ ls
compile  deploy          JOB_TMPDIR  monitor  ondemand  stage
demo    example-text.txt lts_proj    notes    R
[rpb222@sol ~]$ rm example-text.txt
[rpb222@sol ~]$ ls
compile  demo  deploy  JOB_TMPDIR  lts_proj  monitor  notes  ondemand  R  stage
[rpb222@sol ~]$
```

Remove a directory: `rm -r <directory>`

- If your mental map of what you are removing is wrong, you will lose data. **BEWARE!**
- The flag (`-r`) is *recursive*, which means it iterately removes all files and directories nested under the target path.
- Deletes files and folders (and files within the folders ... and so on).
- Some implementations will use a confirmation prompt.
- Override the confirmation with the “force” flag (`rm -rf`).
- To delete only empty directories, use: `rmdir <path>`

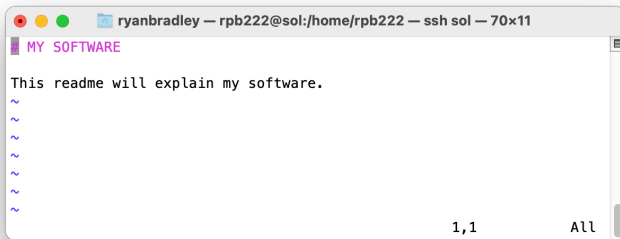
Everything is text

- Text editors are essential to interacting with high-performance computing (HPC) clusters remotely.
- Editors typically include *many* productivity-enhancing functions.
- These functions have an *ethos*.
- We will briefly review the titans of text editors:
 - Vim (vi improved) and Emacs .
- Other editors include: kate , gedit , gvim , nano , kwrite , Sublime

Emacs vs Vim:

- Emacs prefers key combinations.
- Emacs is more graphical.
- Emacs has an entire LISP-based ecosystem, a package manager, and lots of bells and whistles.
- Vim has two modes (insert and command).
- Vim economizes on keypresses, but they can be hard to learn.
- Editorial comment from the author: Vim is superior!
- Both are high-availability.
- Emacs is the default bash mode, see set -o emacs

Vim: commands



```
ryanbradley — rpb222@sol:/home/rpb222 — ssh sol — 70x11
MY SOFTWARE
This readme will explain my software.
~
~
~
~
~
~
~
1,1 All
```

- To enter commands, you must first escape insert mode if necessary (`esc`)
- Then use a colon character (`:`) to enter a command.
- Even if you don't enter a command, you can use simple commands (i.e. `i` , to enter insert mode) from the command mode.
- Save and exit with: `<ESC>` then `:wq`

Vim: insert



```

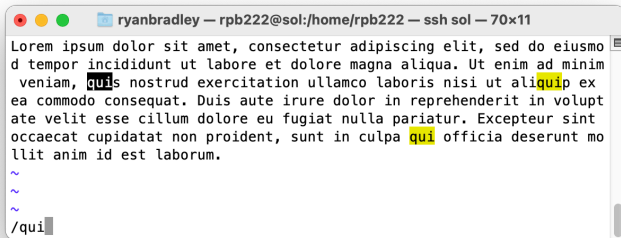
ryanbradley -- rpb222@sol:/home/rpb222 -- ssh sol -- 70x11
MY SOFTWARE
This readme will explain my software.
~
~
~
~
~
~
1,1 All

```

Shortcuts:

- switch to insert mode: `i` , insert at beginning of line: `I`
- append after cursor `a` , append at end of line `A`
- newline after cursor in insert mode `o` , newline before cursor in insert mode `O`
- append at end of line `ea`
- use `dd` to remove a line
- In Emacs and BASH, `ctrl+a` and `ctrl+e` move to the beginning or end of the line.
- In Vim, we use `^` and `$` instead (incidentally this is the regular expression convention).

Vim: searching



```
ryanbradley — rpb222@sol:/home/rpb222 — ssh sol — 70x11
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in reprehenderit in volupt
ate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mo
llit anim id est laborum.
~
~
~
/qui
```

- To search, you must first escape insert mode if necessary (**esc**)
- Enter the search term after a slash: **/mysearch**
- Use **n** and **N** to find the next or previous matches



Vim: cut and paste

```
ryanbradley — rpb222@sol:/home/rpb222/notes — ssh sol — 70x11
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in reprehenderit in volupt
ate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mo
llit anim id est laborum.
~
~
~
-- VISUAL BLOCK --                               1x55      1,55      All
```

- 1 Start in command mode. Move your cursor.
- 2 Type `ctrl+v` and move your cursor to select the text.
- 3 When your selection is complete, type `y` for “yank” i.e. copy.
- 4 Alternately, type `d` for “delete” if you want to “cut” (`dd` deletes a line without a selection).
- 5 Move to a new location.
- 6 Type `p` to paste after the cursor.
- 7 Bonus: what should you type to paste *before* the cursor?

Vim: find and replace

```
ryanbradley — rpb222@sol:/home/rpb222/notes — ssh sol — 70x11
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in reprehenderit in volupt
ate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mo
llit anim id est laborum.
~
~
~
:%s/\. /\.\r/g
```

- 1 Start with some text.
 - 2 Enter command mode.
 - 3 Use the command `:%s/\. /\.\r/g` to globally replace a period followed by a space with a newline instead.
- Use `u` to undo and `r` to redo.
 - Note that we use a carriage return (`\r`) instead of a newline (`\n`).
 - *Warning!* Not all systems use the same newline character (see `dos2unix`) if you encounter this problem.
 - Otherwise, Stack Overflow is your friend.

Vim: block write

```
ryanbradley — rpb222@sol:/home/rpb222/notes — ssh sol — 70x11
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.
~
~
-- VISUAL BLOCK --                               4x1      4,1      All
```

- 1 Use `ctrl+v` and select a column at the beginning of each line.
- 2 Type `shift+I` and then insert a character (for example a hash, `#`, to make a comment in many languages).
- 3 Type `esc, esc` and the character is inserted at the beginning of each line.
- 4 Use `shift + <` or `shift + >` to indent or outdent.

Warning: all tabulators (tabs) are not created equal.

Find

```
tmp — rpb222@sol:/home/rpb222/ortho — ssh sol — 89x18
[rpb222@sol ortho]$ find . | head -n 16
.
./MANIFEST.in
./LICENSE.txt
./setup.cfg
./ortho
./ortho/logs.py
./ortho/yaml_trestle.py
./ortho/git.py
./ortho/yaml_include.py
./ortho/reexec.py
./ortho/terminal_view.py
./ortho/utills.py
./ortho/locker.py
./ortho/diagnose.py
./ortho/stage.py
./ortho/dotdict.py
[rpb222@sol ortho]$
```

- Search the filesystem with `find <path>`
- Filter names with `find <path> -name "*phrase*"`
- Search directories with `find . -type d`
- Search files with `find . -type f`

Input-Output (I/O) Redirection

- 1 Three metaphorical files serve as I/O streams.
 - `stdin` standard input
 - `stdout` standard output (1)
 - `stderr` standard error (2)
- 2 Many applications can be connected by these streams.
 - `<` writes a file to `stdin` on an application
 - `>` writes `stdout` to a file
 - `>>` appends `stdout` to a file
 - `|` connects `stdout` of one application to `stdin` of another

Send a stream to a file

```
tmp — rpb222@sol:/home/rpb222/notes — ssh sol — 89x18
[rpb222@sol notes]$ echo -e "Hello, World" > myfile.txt
[rpb222@sol notes]$ cat myfile.txt
Hello, World
[rpb222@sol notes]$ cat myfile.txt > another-file.txt
[rpb222@sol notes]$ cat another-file.txt
Hello, World
[rpb222@sol notes]$ cat myfile.txt >> another-file.txt
[rpb222@sol notes]$ cat another-file.txt
Hello, World
Hello, World
[rpb222@sol notes]$
```

- Echo a string into a file `echo -e "Hello, World" > myfile.txt`
- Use the concatenate (`cat`) command to stream a file to `stdout` with `cat myfile.txt`
- Store the results of find in a file with `find . -name "*git*" > my-search.txt`

Regular expressions

```

tmp — rpb222@sol:/home/rpb222/ortho — ssh sol — 89x18
[rpb222@sol ortho]$ find . -type f | grep .py | head -n 10
./ortho/logs.py
./ortho/yaml_trestle.py
./ortho/git.py
./ortho/yaml_include.py
./ortho/reexec.py
./ortho/terminal_view.py
./ortho/utills.py
./ortho/locker.py
./ortho/diagnose.py
./ortho/stage.py
[rpb222@sol ortho]$

```

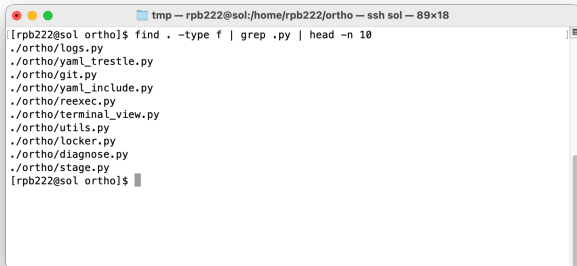
- In this example, we use `find` to traverse a directory tree and `grep` to search for Python files.
- Keep it simple, use `find . -name <string>` instead.
- Unlock better searching with *extended* regular expressions.

```

$ find . -type f | egrep ".*\[^\_][^/]*.py$"
./ortho/logs.py
./ortho/git.py
./ortho/reexec.py

```

Save to a file



```
tmp -- rpb222@sol:/home/rpb222/ortho -- ssh sol -- 89x18
[rpb222@sol ortho]$ find . -type f | grep .py | head -n 10
./ortho/logs.py
./ortho/yaml_trestle.py
./ortho/git.py
./ortho/yaml_include.py
./ortho/reexec.py
./ortho/terminal_view.py
./ortho/utils.py
./ortho/locker.py
./ortho/diagnose.py
./ortho/stage.py
[rpb222@sol ortho]$
```

- You can stream the results of a command into a file to save the results for later.
- The combinations are endless.

```
$ find . -type f | egrep ".*\[^\_][^/]*.py$" > results.txt
$ head results.txt
$ cat results.txt | head
./logs.py
./git.py
./reexec.py
```


Reading a permission string

```

tmp — rpb222@sol:/home/rpb222/notes — ssh sol — 104x19
[rpb222@sol notes]$ ls -all
total 20
drwxr-xr-x  2 rpb222 faculty 4096 Feb  7 06:45 .
drwx----- 23 rpb222 faculty 4096 Feb  7 07:12 ..
-rw-r--r--  1 rpb222 faculty  26 Feb  7 05:27 another-file.txt
-rw-r--r--  1 rpb222 faculty  13 Feb  6 20:35 myfile.txt
-rw-r--r--  1 rpb222 faculty  13 Feb  7 06:45 mytimes
[rpb222@sol notes]$ chmod g+w myfile.txt
[rpb222@sol notes]$ ls -all
total 20
drwxr-xr-x  2 rpb222 faculty 4096 Feb  7 06:45 .
drwx----- 23 rpb222 faculty 4096 Feb  7 07:12 ..
-rw-r--r--  1 rpb222 faculty  26 Feb  7 05:27 another-file.txt
-rw-rw-r--  1 rpb222 faculty  13 Feb  6 20:35 myfile.txt
-rw-r--r--  1 rpb222 faculty  13 Feb  7 06:45 mytimes
[rpb222@sol notes]$ █

```

- The permission string distinguishes directories from files.
- The string uses groups of three read-write-execute characters.
- These three groups mark the permissions for the owner (user), group, and all other users.
- Execute permissions are required to change (`cd`) into a directory.
- Execute permissions are required to run binaries or scripts directly (without an interpreter).
- Change permissions with `chmod` .
- Permissions codes (an octal number) provide a shortcut.

More metadata information



```
tmp -- rpb222@sol:/home/rpb222/notes -- ssh sol -- 89x18
[rpb222@sol notes]$ stat another-file.txt
  File: another-file.txt
  Size: 26          Blocks: 8          IO Block: 1048576 regular file
Device: 48h/72d Inode: 353896923  Links: 1
Access: (0644/-rw-r--r--)  Uid: (300318/  rpb222)   Gid: ( 355/  faculty)
Access: 2023-02-07 05:27:34.802658787 -0500
Modify: 2023-02-07 05:27:34.802658787 -0500
Change: 2023-02-07 05:29:29.078677879 -0500
 Birth: -
[rpb222@sol notes]$
```

- Use `stat <file>` to review all of the metadata.
- On MacOS use `stat -x <file>` for the same format.

Every program has a number

```

[rpb222@sol ortho]$ ps
  PID TTY          TIME CMD
3595422 pts/1    00:00:00 bash
3820523 pts/1    00:00:00 ps
[rpb222@sol ortho]$ ps xao pid,user,command | grep -v root | head
  PID USER          COMMAND
 1199 rpc            /usr/bin/rpcbind -w -f
 1244 daemon        /usr/sbin/rngd -f --fill-watermark=0 -x pkcs11 -x nist -x qrypt -D daemo
n:daemon
 1246 chrony       /usr/sbin/chronyd
 1247 polkitd      /usr/lib/polkit-1/polkitd --no-debug
 1250 dbus         /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
systemd-activation --syslog-only
 1372 munge        /usr/sbin/munged
 1444 mysql        /usr/libexec/mysqld --basedir=/usr
 1950 postfix      qmgr -l -t unix -u
 2142 slurm        /usr/local/slurm-21.08.8-2/sbin/slurmdbd -D -s
[rpb222@sol ortho]$

```

- Each program has a process identifier (**PID**).
- The **ps** command tells you the processes in the current shell.
- You can also scan other processes to monitor your system.

```

# use ps to review the process table by PID
$ ps xao pid,user,command | grep -v root | head

```

Running in the Background

```

tmp — rpb222@sol:/home/rpb222/notes — ssh sol — 89x18
[rpb222@sol notes]$ (while true; do sleep 4; date +%Y%m%d%H%M > mytimes; done) &
[1] 3876521
[rpb222@sol notes]$ jobs
[1]+  Running                  ( while true; do
    sleep 4; date +%Y%m%d%H%M > mytimes;
done ) &
[rpb222@sol notes]$ kill %1
[rpb222@sol notes]$ tail mytimes
202302070645
[1]+  Terminated              ( while true; do
    sleep 4; date +%Y%m%d%H%M > mytimes;
done )
[rpb222@sol notes]$ █

```

- To run commands in the background, use the ampersand (`&`).
- Use `jobs` to list the index and PIDs for background processes in this shell.
- Resume a job in the foreground by index with `fg <N>`
- Resume a job in the background by index with `bg <N>`
- Terminate a job by index with `kill %<N>` or by PID with `kill <N>`
- Use `nohup` to detach a program from the background (advanced example).

Interrupts

- Review system activity with `top` or `htop`
- The `htop` program lets you select a program and send it a signal.
- Use `ctrl+c` to send an `INT` or interrupt signal to kill a foreground process.
- Use `ctrl+z` to send a `SIGTSTP` or stop signal to pause foreground process.
- Use `ctrl+d` to send an end-of-file `EOF` *character* which closes your terminal.

The Environment is a state

```

tmp -- rpb222@sol:/home/rpb222/notes -- ssh sol -- 104x19
[rpb222@sol notes]$ env | head -n 1
LD_LIBRARY_PATH=/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/py-pip/20.2-ig2u7ky/lib:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/py-setuptools/50.3.2-riobjp7/lib:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/python/3.8.6-lesvst/lib:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/intel-2021.3.0/mvapich2/2.3.4-7drzz4/lib:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/lib/x64:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/lib/emu:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/compiler/lib/intel64_lin:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/lib
[rpb222@sol notes]$ MYNAME=JOHN
[rpb222@sol notes]$ echo $MYNAME
JOHN
[rpb222@sol notes]$ export MYNAME=JOHN
[rpb222@sol notes]$ bash
manpath: warning: $MANPATH set, ignoring /etc/man_db.conf
System Architecture: avx2
Processor Family: haswell
[rpb222@sol notes]$ echo $MYNAME
JOHN
[rpb222@sol notes]$ █

```

- Use the `env` command to list all variables.
- Define variables for *this shell only*.
- To use variables in a subshell, export them via `export MYNAME="JOHN"`
- Environment variables follow a convention that tells a compiler or the kernel where to find other software.
- Therefore, the environment holds a *hidden configuration* or *state* of the system.

Custom variables

```
ryanbradley -- rpb222@sol:/home/rpb222 -- ssh sol -- 104x19
[rpb222@sol ~]$ code_place=$HOME/ortho
[rpb222@sol ~]$ echo $code_place
/home/rpb222/ortho
[rpb222@sol ~]$ echo "our code is at $code_place/ortho/reexec.py"
our code is at /home/rpb222/ortho/ortho/reexec.py
[rpb222@sol ~]$ code_path=$code_place/ortho/reexec.py
[rpb222@sol ~]$ ls $code_path
/home/rpb222/ortho/ortho/reexec.py
[rpb222@sol ~]$ echo "our code is at $code_path"
our code is at /home/rpb222/ortho/ortho/reexec.py
[rpb222@sol ~]$
[rpb222@sol ~]$ echo $PATH
/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/py-pip/20.2-ig2u7ky/bin:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/py-setuptools/50.3.2-riojq7/bin:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/gcc-8.3.1/python/3.8.6-lesvfst/bin:/share/Apps/lusoft/opt/spack/linux-centos8-haswell/intel-2021.3.0/mvapich2/2.3.4-7drazz4/bin:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/bin/intel64:/share/Apps/intel-oneapi/2021/compiler/2021.3.0/linux/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/rpb222/bin
[rpb222@sol ~]$
```

- Custom variables can be defined and reused.
- You can echo variables to standard output.
- You can compose new variables from existing ones.
- You can use variables as arguments.
- Many variables with specific names are used by compilers and the operating system.

Startup

- Whenever you log into a *NIX computer, shell scripts are automatically loaded.
- In BASH, we load:
 - `/etc/profile` (login terminal only)
 - `/etc/bashrc`
 - `$HOME/.bash_profile` (login terminal only)
 - `$HOME/.bashrc`
- Use the profile for user-facing tools, for example `alias` .
- Anything that subshells should inherit should be placed in the `bashrc` .
- Other shells use other files, for example on MacOS `$HOME/.zshrc` .



Loops and Conditionals

```
tmp — rpb222@sol:/home/rpb222/notes — ssh sol — 102x17
[rpb222@sol notes]$ if [ -f myfile.txt ]; then echo "my file exists!"; else "missing!"; fi
my file exists!
[rpb222@sol notes]$ export PI_APPROX=3.14
[rpb222@sol notes]$ if [ -z "$PI" ]; then echo "PI is not defined"; fi
PI is not defined
[rpb222@sol notes]$ if [ ! -z "$PI_APPROX" ]; then echo "we sort of know pi: ${PI_APPROX}"; fi
we sort of know pi: 3.14
[rpb222@sol notes]$ for i in $(seq 1 5); do echo "counting: $i"; done
counting: 1
counting: 2
counting: 3
counting: 4
counting: 5
[rpb222@sol notes]$ █
```

BASH is a fully-featured programming language

Arrays

```

tmp — rpb222@sol:/home/rpb222/ortho/ortho — ssh sol — 89x18
[rpb222@sol ortho]$ ls *.py
bash.py      functional.py  __main__.py    test_dispatch.py  utils.py
cli.py       git.py        metadata.py    test_trestle.py  yaml_compositor.py
diagnose.py  __init__.py  reexec.py     test_yaml_builder.py  yaml_include.py
dispatch.py  locker.py    stage.py      test_yaml.py     yaml.py
dotdict.py   logs.py      terminal_view.py  text_viewer.py   yaml_trestle.py
[rpb222@sol ortho]$ var=$(ls *.py)
[rpb222@sol ortho]$ echo $var
bash.py cli.py diagnose.py dispatch.py dotdict.py functional.py git.py __init__.py locker
.py logs.py __main__.py metadata.py reexec.py stage.py terminal_view.py test_dispatch.py
test_trestle.py test_yaml_builder.py test_yaml.py text_viewer.py utils.py yaml_compositor
.py yaml_include.py yaml.py yaml_trestle.py
[rpb222@sol ortho]$ for i in ${var[@]}; do echo $i; done
bash.py
cli.py
diagnose.py
dispatch.py
dotdict.py

```

- The BASH shell has many ways to work with lists.
- But they can be a bit more work than a traditional programming language (this is the tradeoff for high-availability).
- Make a simple **for** loop with a special array syntax that uses spaces to delimit the array items.

```
$ for i in ${var[@]}; do echo $i; done
```

Make a script

First, we will learn a new trick. BASH can replace spaces with underscores using *regular expressions*, a powerful text-manipulation standard that is available across many programming languages.

```
$ x="This sentence has words"  
$ echo "${x// /_}"  
This_sentence_has_words
```

Next, prepare a list of filenames with spaces, using `touch`, and quotes. This is your test set.

Assignment: write a BASH script with a `for` loop that renames the files and replaces spaces with underscores. Start with the following BASH script. Remember to use `chmod` so you can execute it with `./myscript.sh` once it's ready.

Closing

QUESTIONS?

- 1 Learning linux is a logarithmic process. You can get a lot of milage out of the basics, and you can use it for a long time before using the most advanced features.
- 2 Read the factory manual! Use the `manpages` !
- 3 Send me a message to learn more: `rpb222@lehigh.edu` .