LEHIGH UNIVERSITY

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?

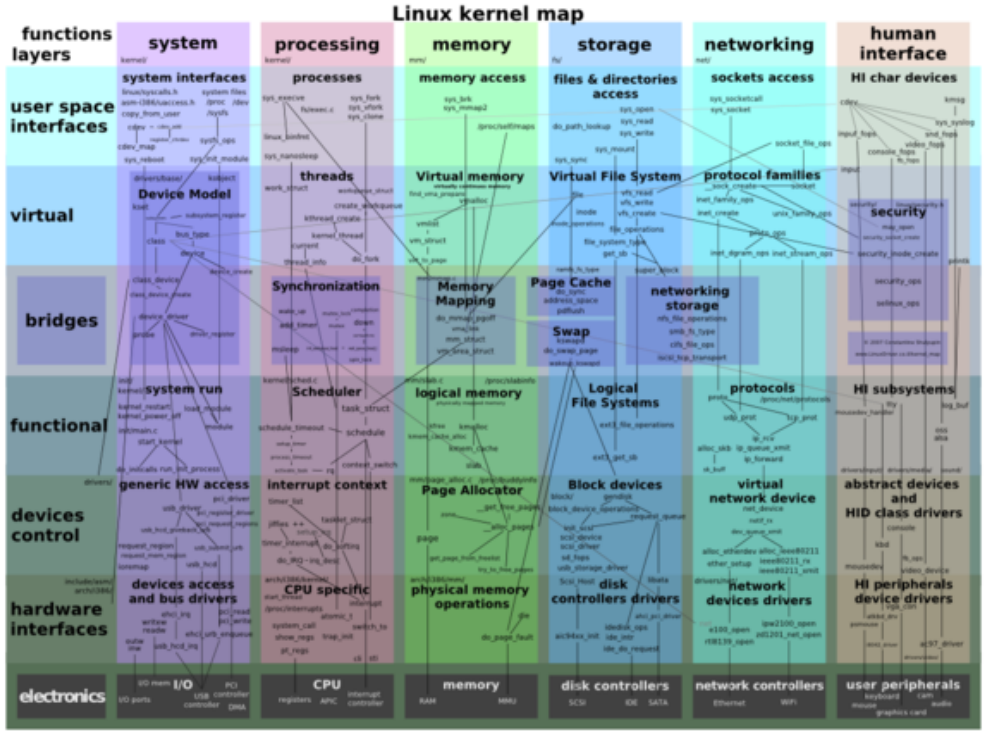


Figure 1: Map of the kernel, via Wikipedia.

‘

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

LEHIGH UNIVERSITY

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

LEHIGH UNIVERSITY

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?

LEHIGH UNIVERSITY

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` , `csh` , `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

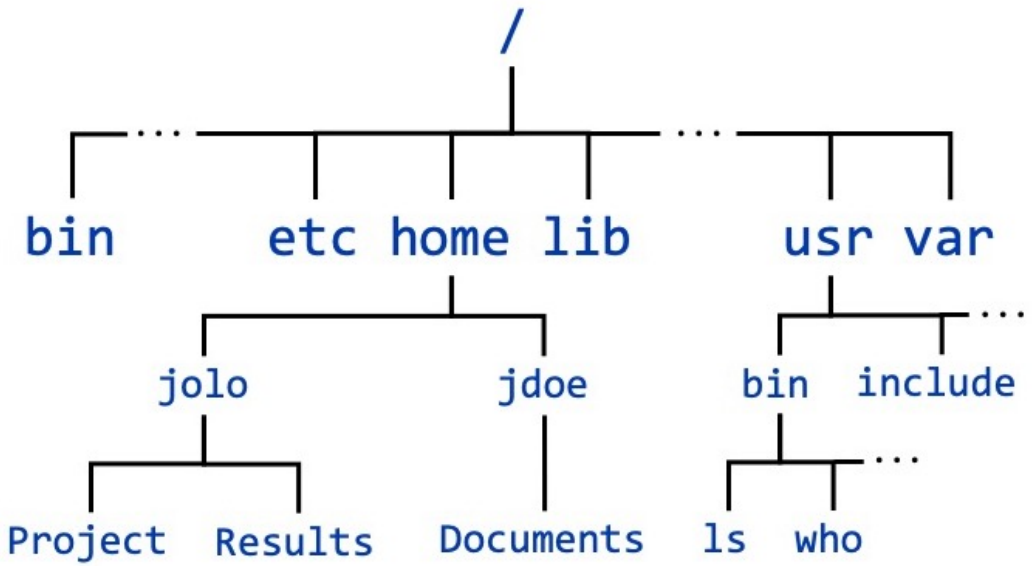
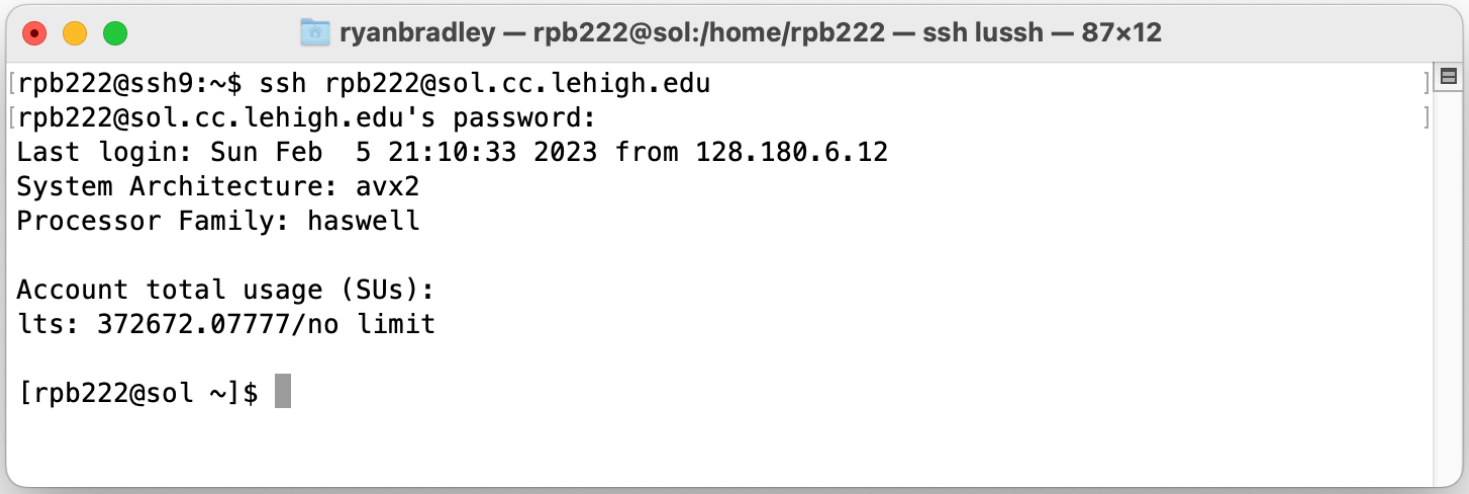


Figure 3: Example filesystem (via Cornell Virtual Workshop).

- 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

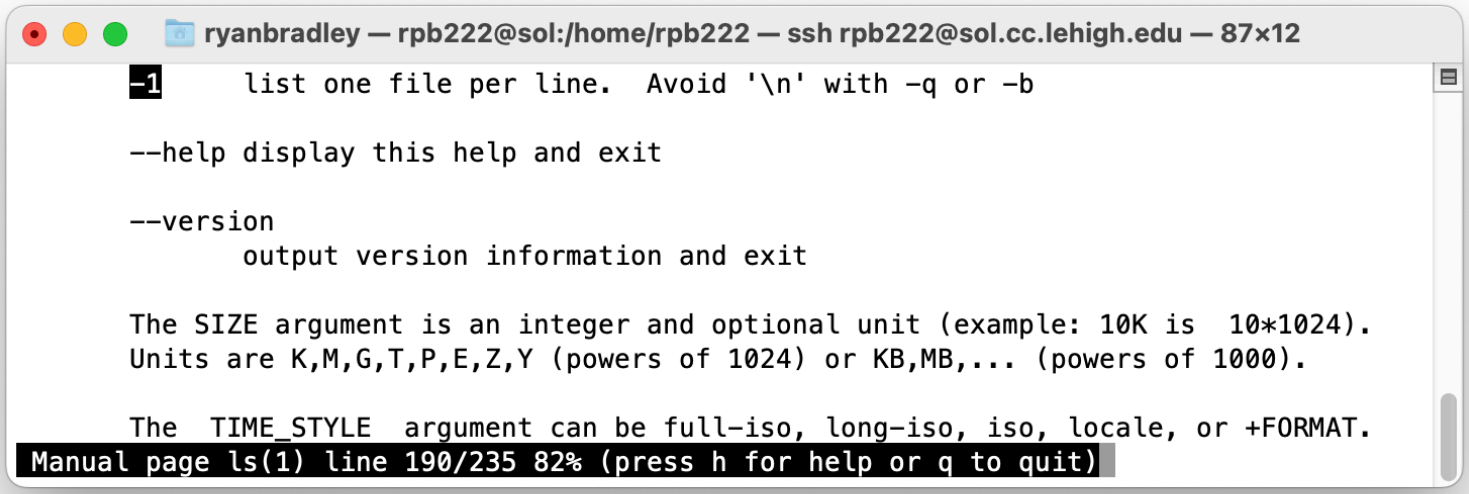


Use `ssh` to log on to `sol` with this command:

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

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

Read the manual



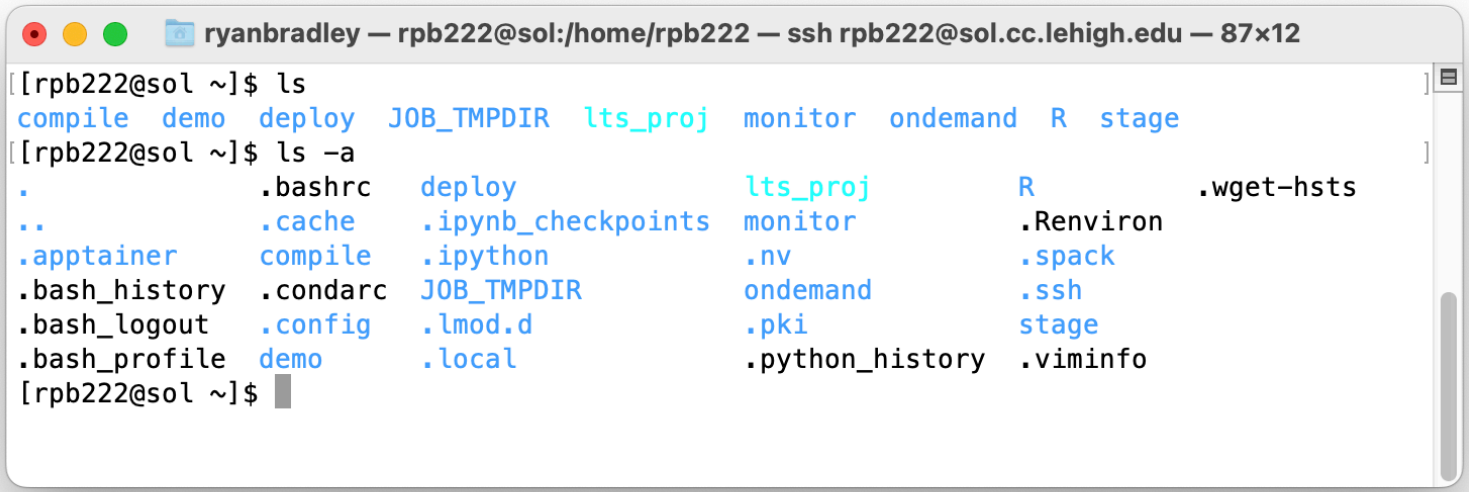
Reveal the “manpages” with: `man ls`

- *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

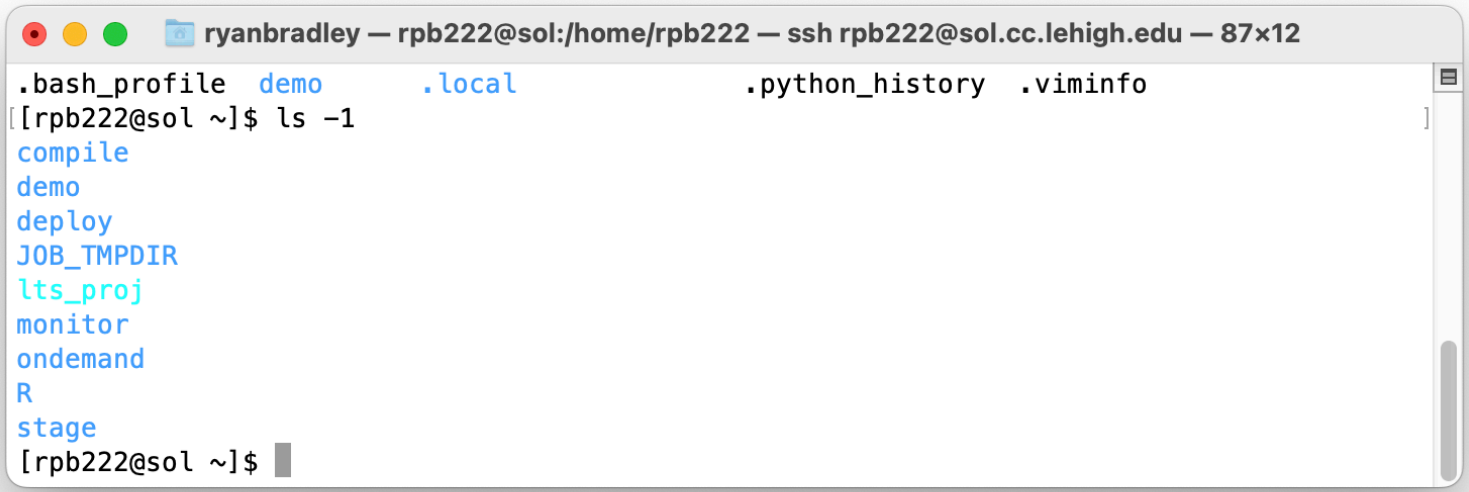
List files: `ls <path?>`

List files



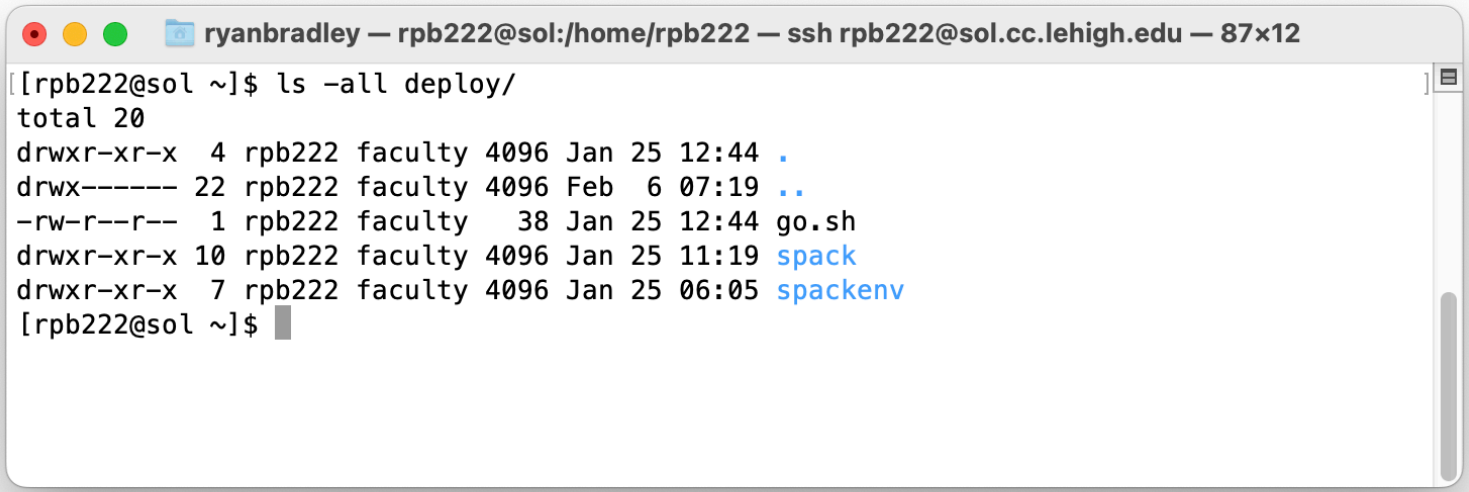
List hidden (“all”) files: `ls -a <path?>`

List files



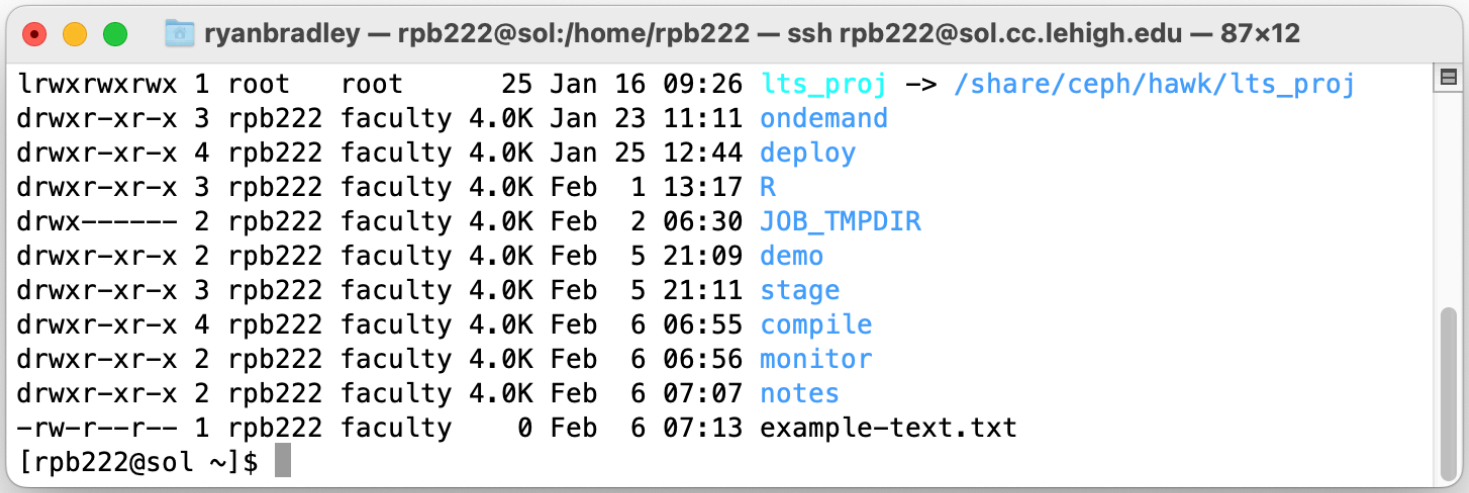
List files in a column: `ls -1 <path?>`

List files



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

List files

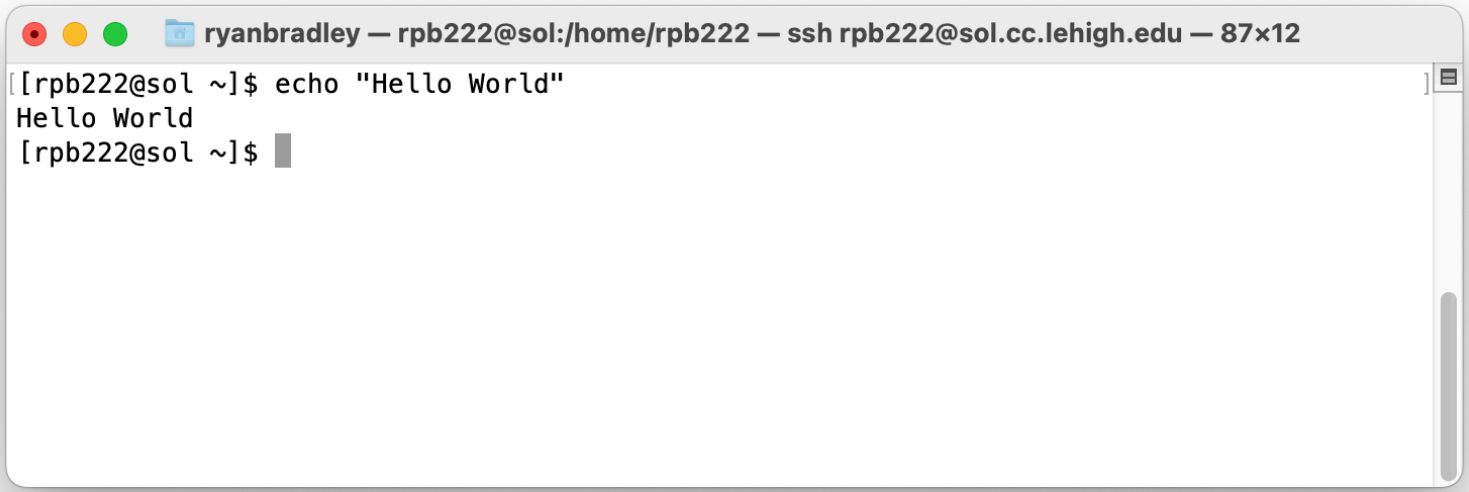


Sort the list by descending modification time:

`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



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

Echo your location in the filesystem: `pwd`

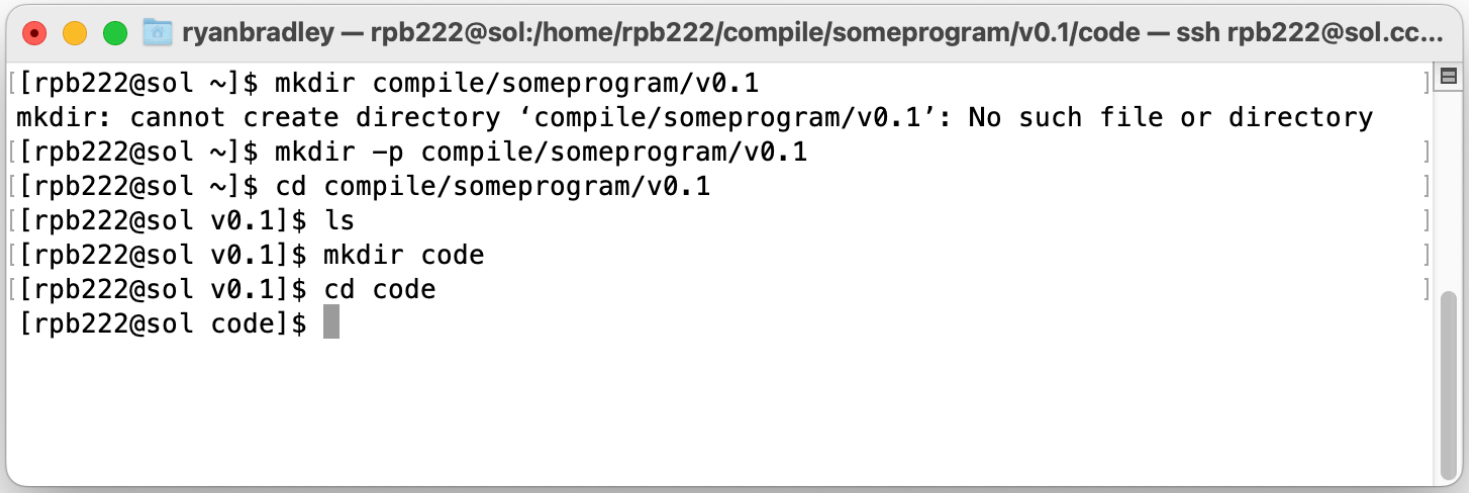
- Sometimes called “current working directory” in other contexts.
- We can also ask an “environment variable” to tell us this (more on this later).
- Usually included in the `$PS1`, or the prompt on the left.
- The `~` symbol means home (a.k.a. `$HOME`).

Change directory

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

- Without an argument, you return to “home”, typically `/home/<username>` .
- The argument can me 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.

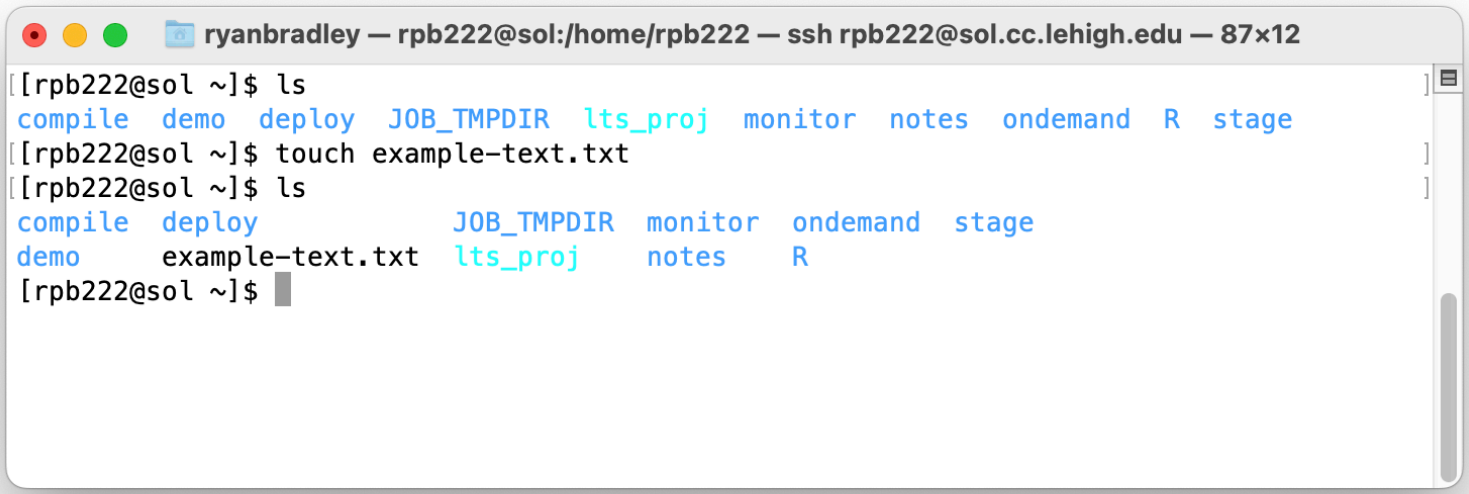
Make a directory



Make a directory: `mkdir <dirname>`

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

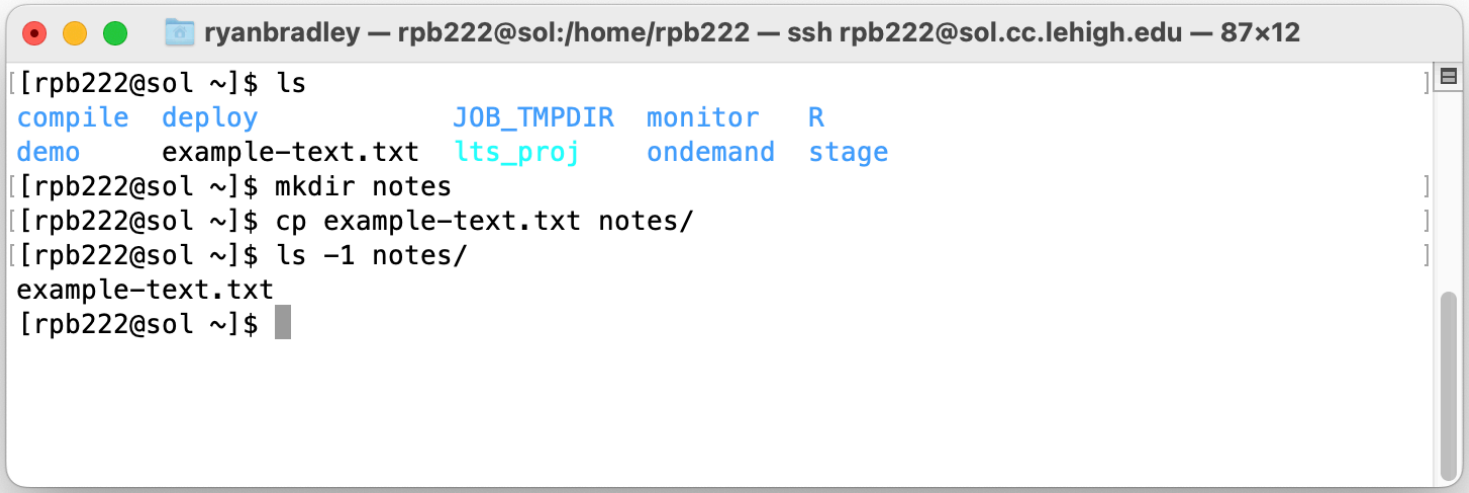
Create a file



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

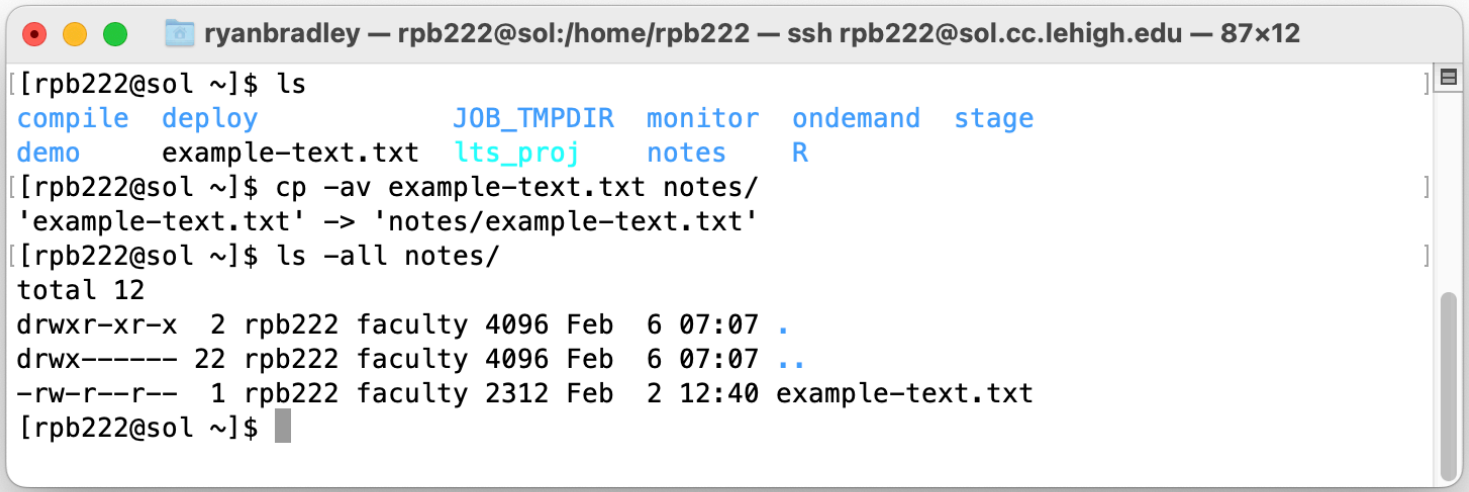


The copy commands duplicates a file:

```
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



Maintain metadata ("archive"):

```
cp -av <source> <destination>
```

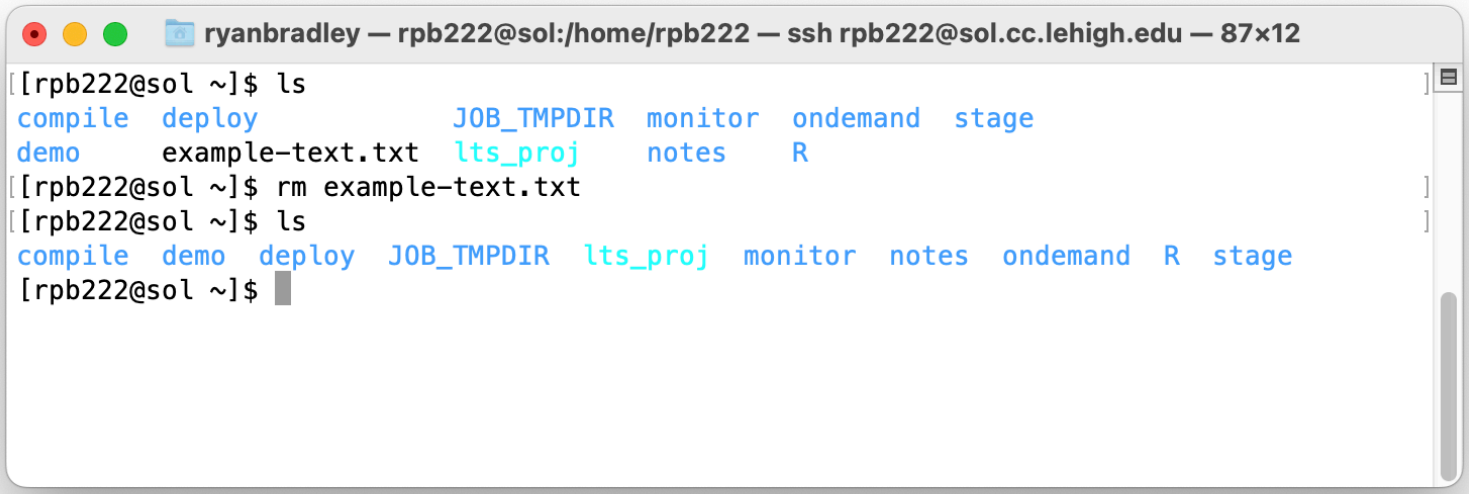
- When you copy a file, it "creates" a duplicate with a timestamp for "now".
- To preserve the metadata, use the "archive" feature, which preserves the creation and modification times, as well as other metadata (owner, permissions, etc).
- Flags are explained in the manpages.

Rename a file

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.

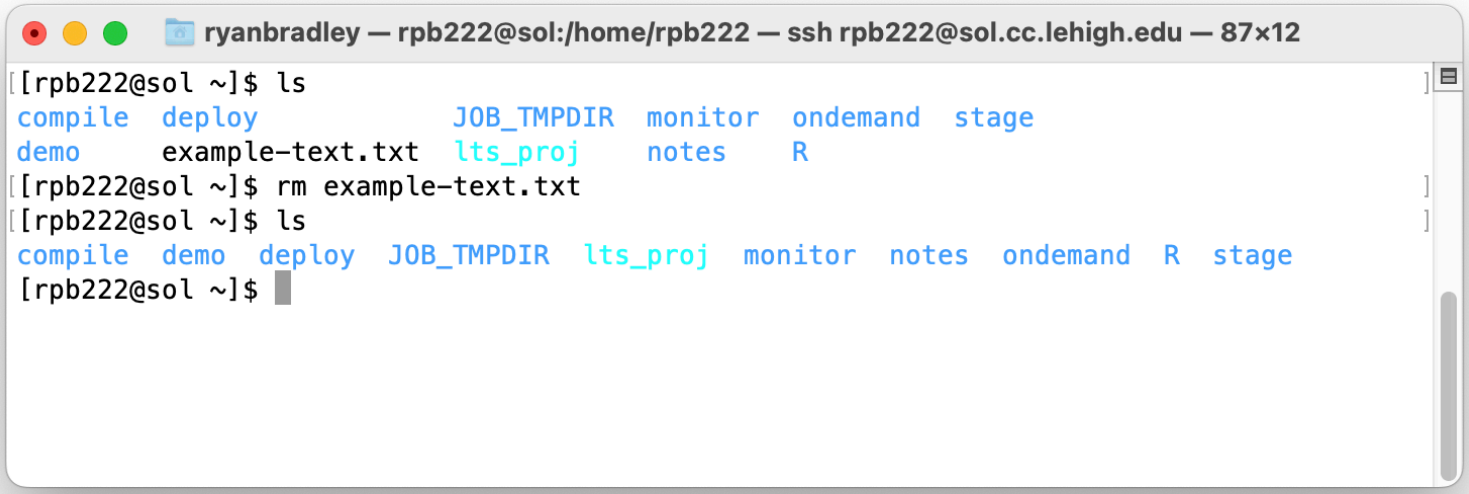
Remove



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



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>`

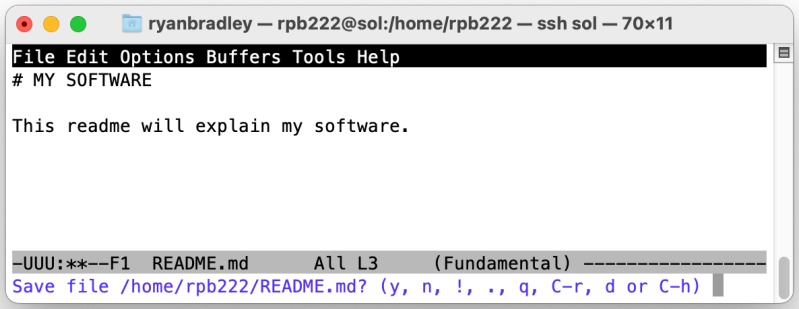
Everything is text

- 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:
  $\longrightarrow$ `Vim` (`vi improved`) and `Emacs` .
- Other editors include: `kate` , `gedit` , `gvim` , `nano` , `kwrite` , `Sublime`

Emacs vs Vim:

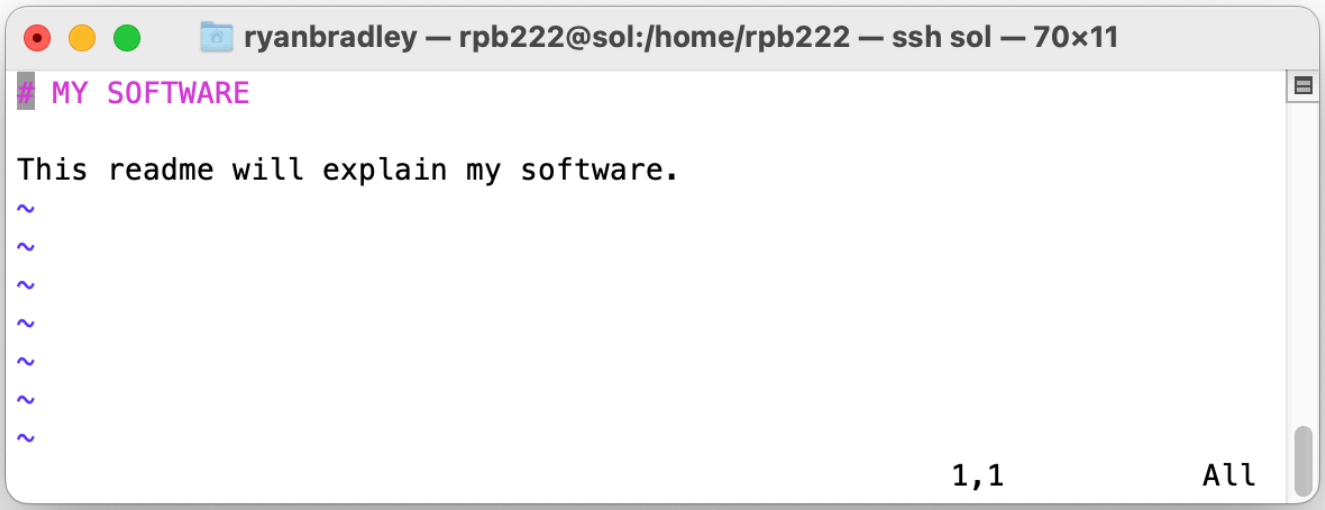
- Emacs prefers key combinations.
- Emacs is more graphical.
- Emacs has an entire LISP-based ecosystem, a package manager, and lots of bells and whistles.
- Vim has two modes (insert and command).
- Vim economizes on keypresses, but they can be hard to learn.
- Editorial comment from the author: Vim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

Emacs demo



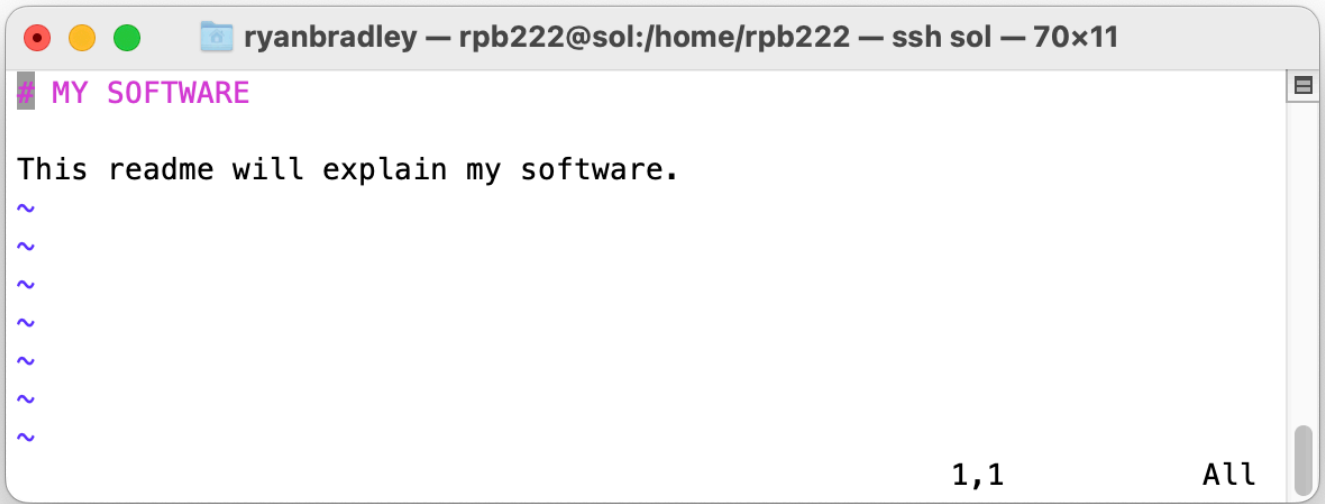
- What is a buffer? What is a minibuffer?
- Command syntax in most documentation uses `ctrl` and `meta` (alt) in combination with other keys.
- For example `C-x C-c` means “press and hold `ctrl` while pressing `x`, then press and hold `ctrl` while pressing `c`” (you do not have to release `ctrl` between commands).
- Common commands:
  - `C-x C-c` exit (it asks if you want to save)
  - `M-w` copy selected text
  - `C-y` pasted selected text
- Use `q` and/or `ESC` to escape the minibuffer.

Vim: commands



- 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`

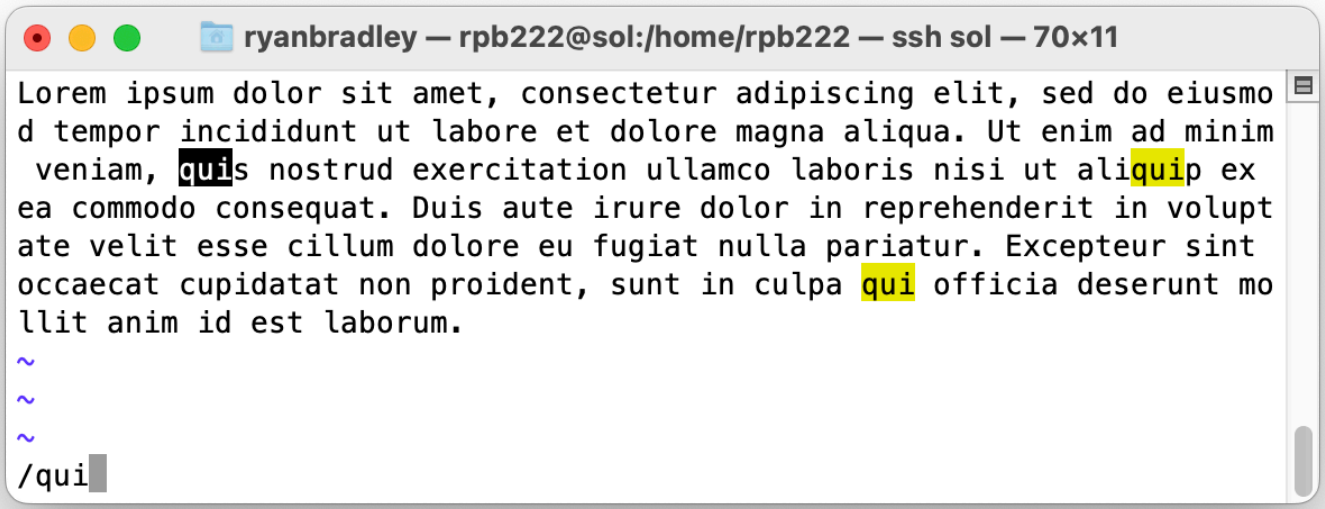
Vim: insert



Shortcuts:

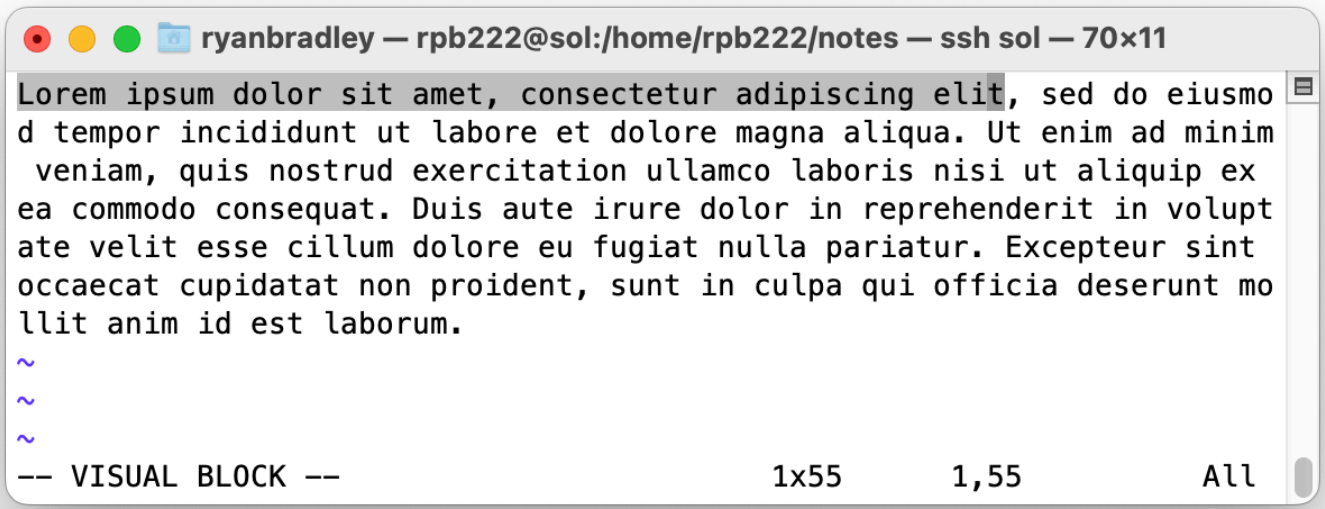
- switch to insert mode: `i` , insert at beginning of line: `I`
- append after cursor `a` , append at end of line `A`
- newline after cursor in insert mode `o` , newline before cursor in insert mode `O`
- append at end of line `ea`
- use `dd` to remove a line
- In Emacs and BASH, `ctrl+a` and `ctrl+e`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

Vim: searching



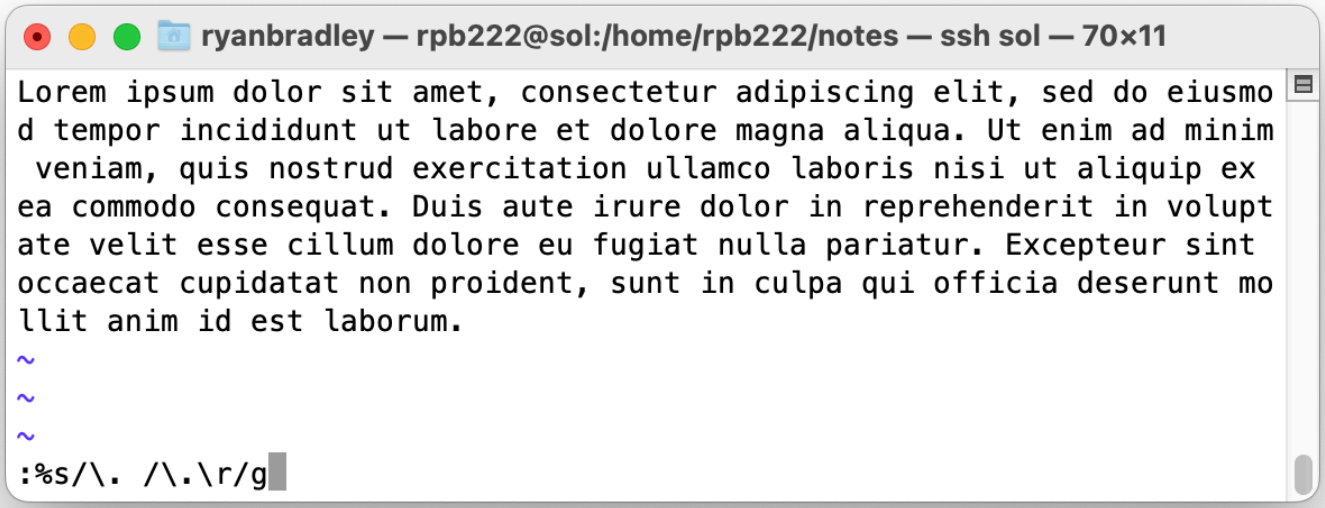
- To search, you must first escape insert mode if necessary (`esc`)
- Enter the search term after a slash: `/mysearch`
- Use `n` and `N` to find the next or previous matches

Vim: cut and paste



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?

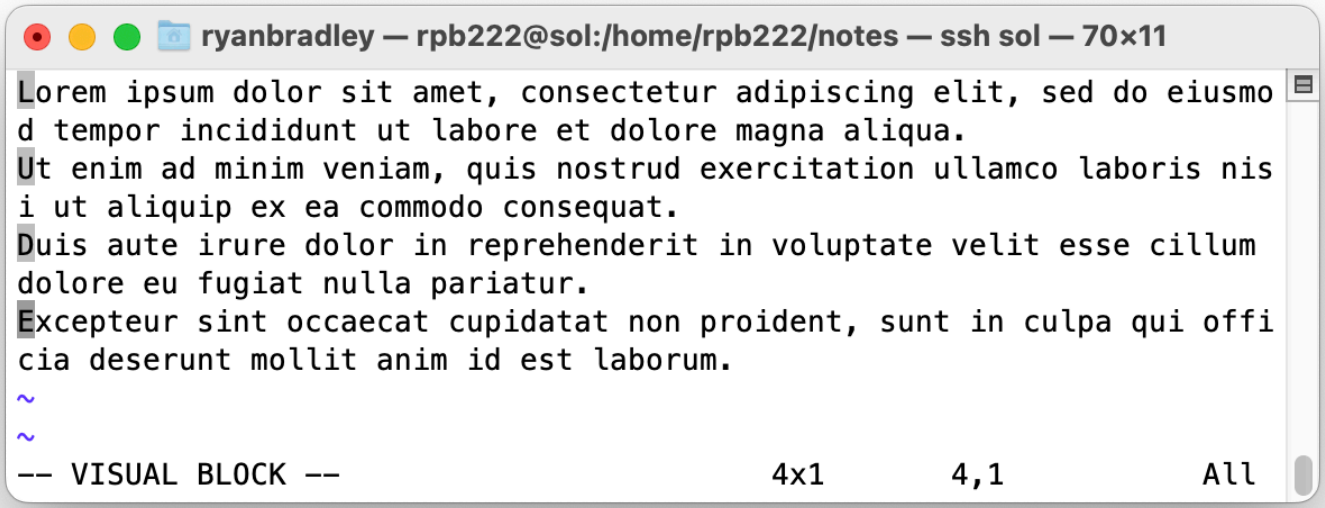
Vim: find and replace



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.

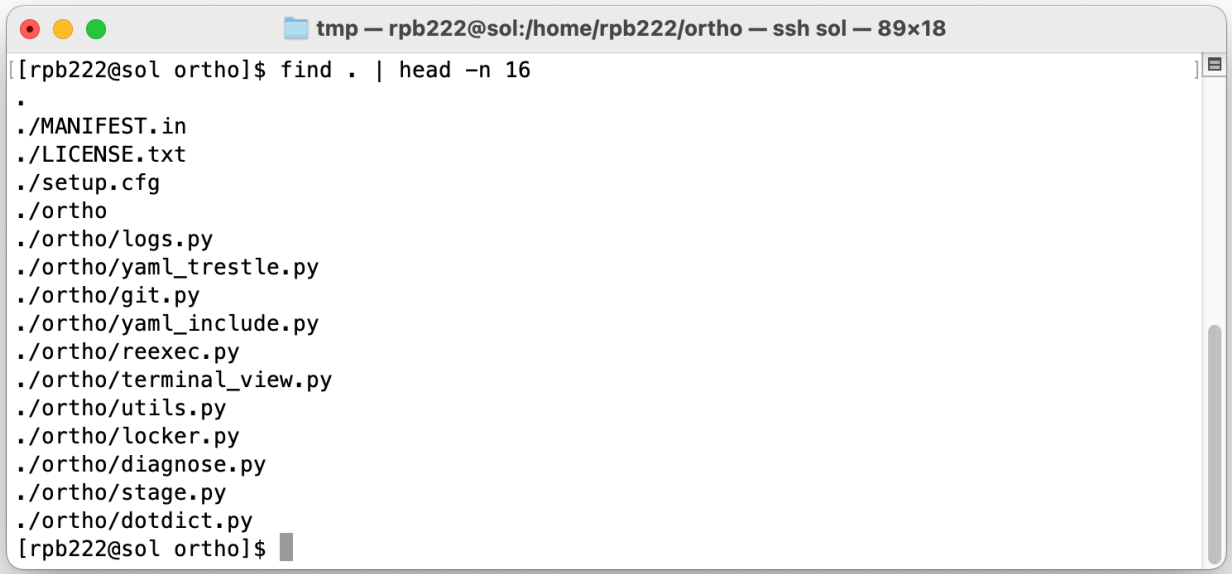
Vim: block write



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

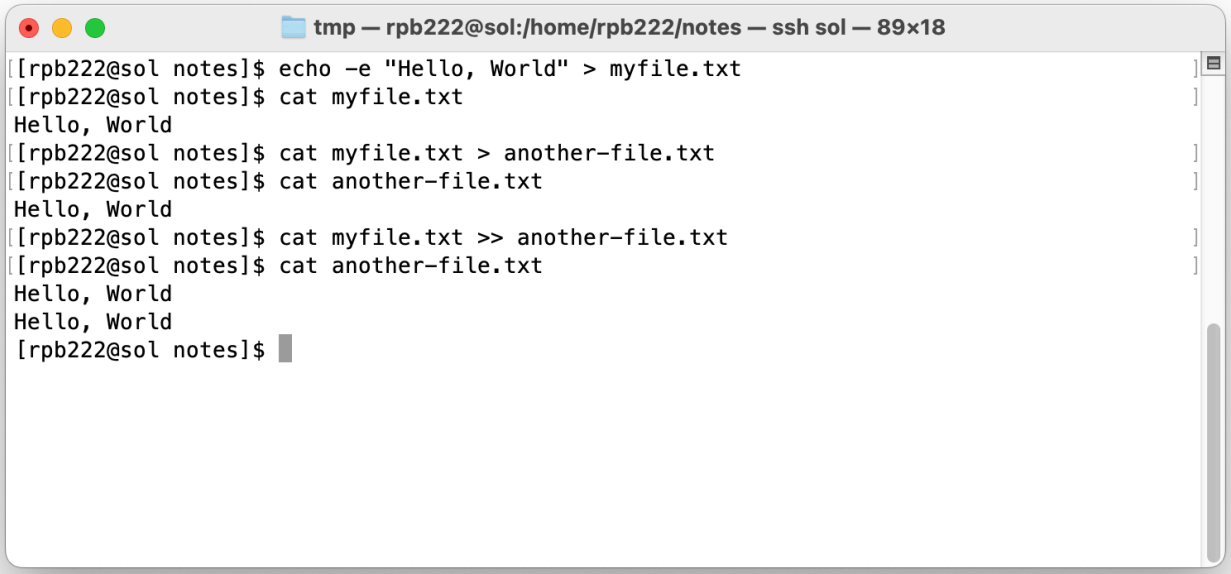


- Search the filesystem with `find <path>`
- Filter names with `find <path> -name "*phrase*"`
- Search directories with `find . -type d`
- Search files with `find . -type f`

Input-Output (I/O) Redirection

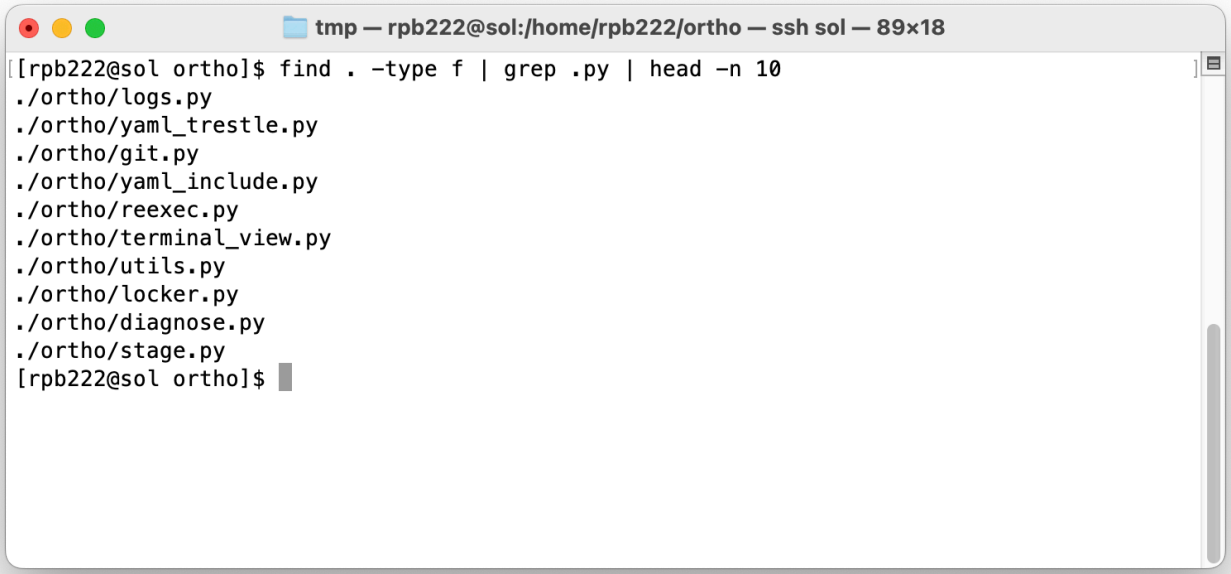
1. Three metaphorical files serve as I/O streams.
   - `stdin` standard input
   - `stdout` standard output (1)
   - `stderr` standard error (2)
2. Many applications can be connected by these streams.
   - `<` writes a file to `stdin` on an application
   - `>` writes `stdout` to a file
   - `>>` appends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

Send a stream to a file



- 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`

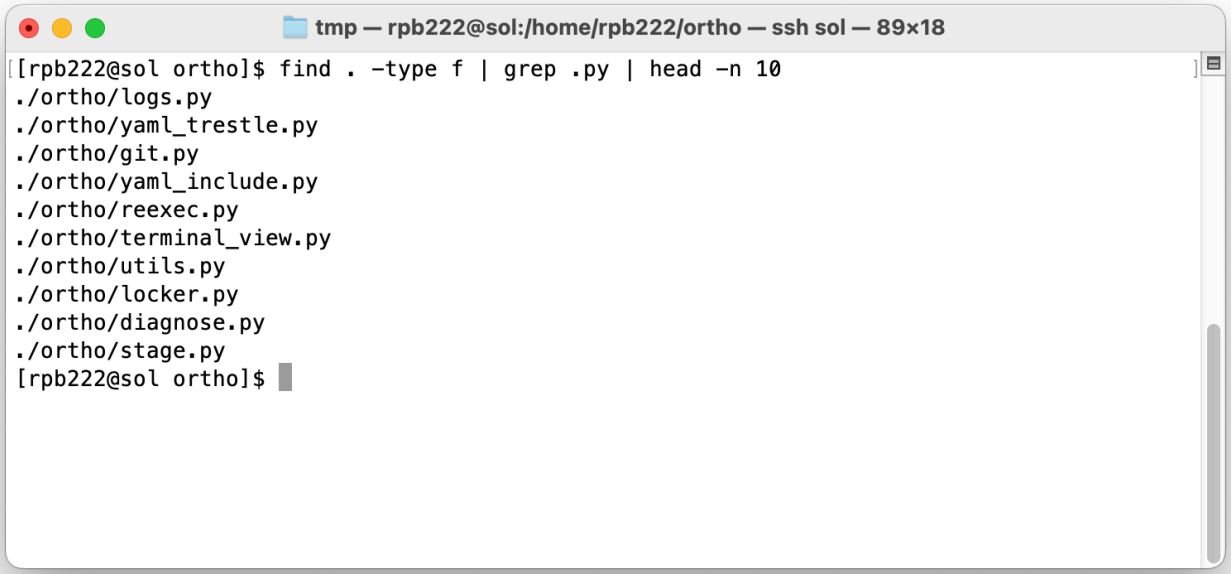
Regular expressions



- 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
```

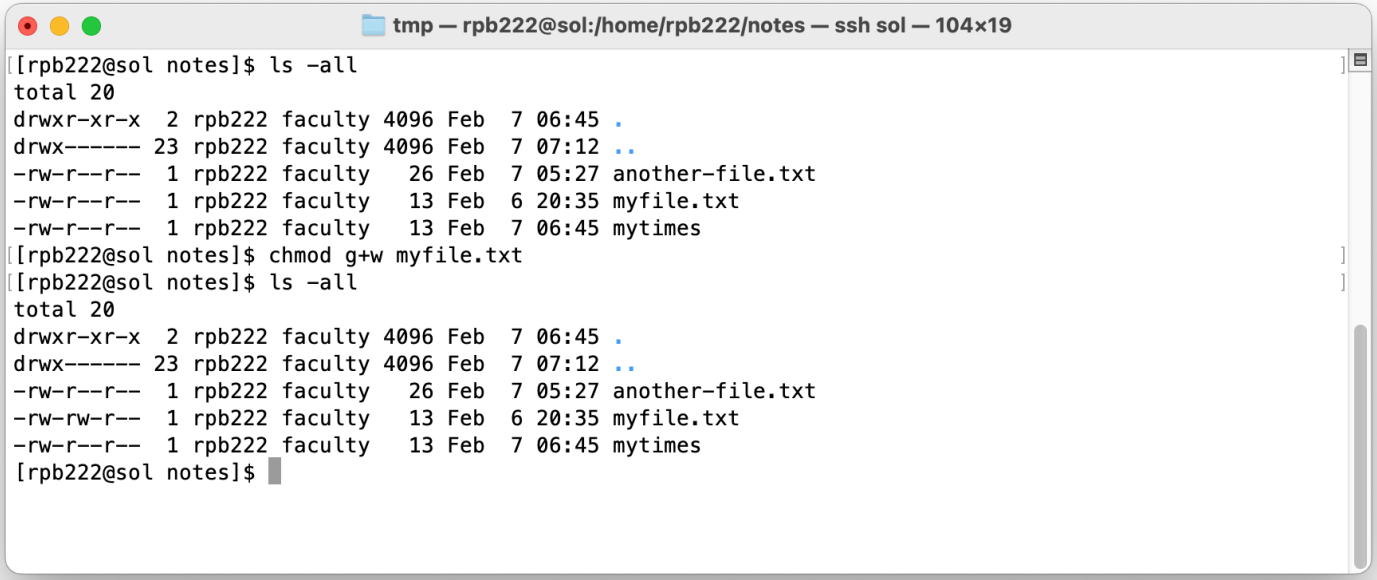
Save to a file



- 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



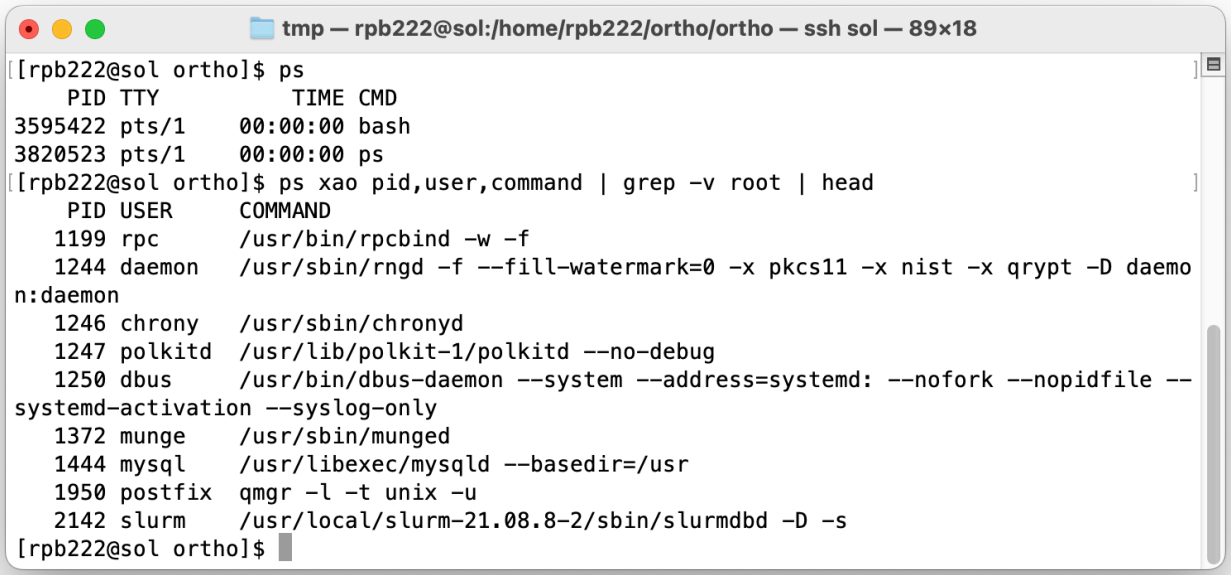
- The permission string distinguishes directories from files.
- The string uses groups of three read-write-execute characters.
- These three groups mark the permissions for the owner (user), group, and all other users.
- Execute permissions are required to change (`cd`) into a directory.
- Execute permissions 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

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

LEHIGH UNIVERSITY

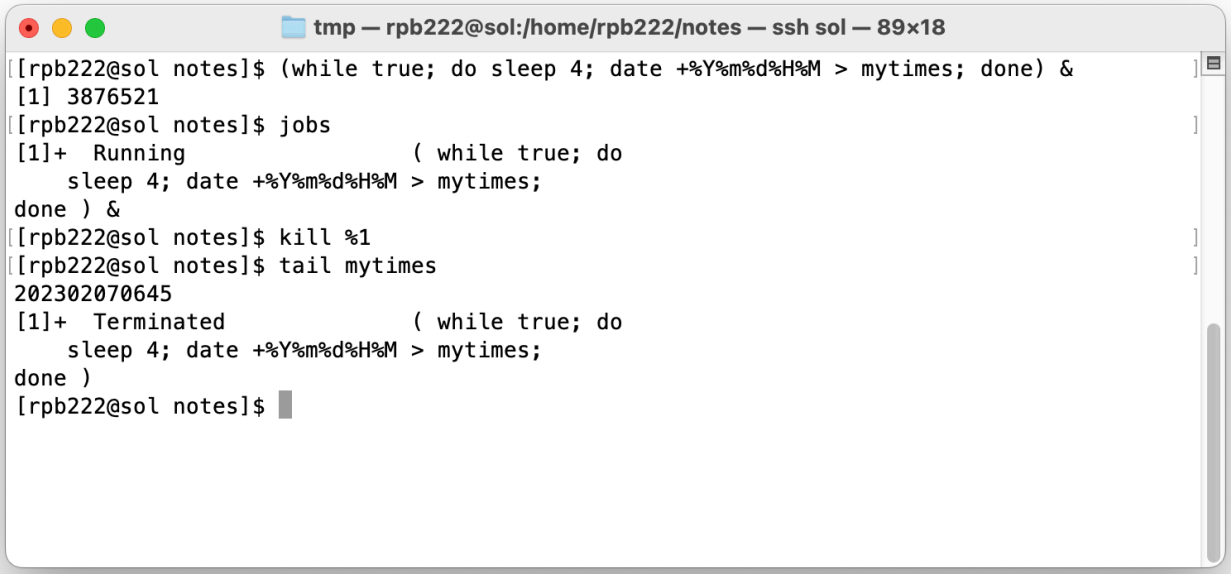
Every program has a number



- 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 Backgroud

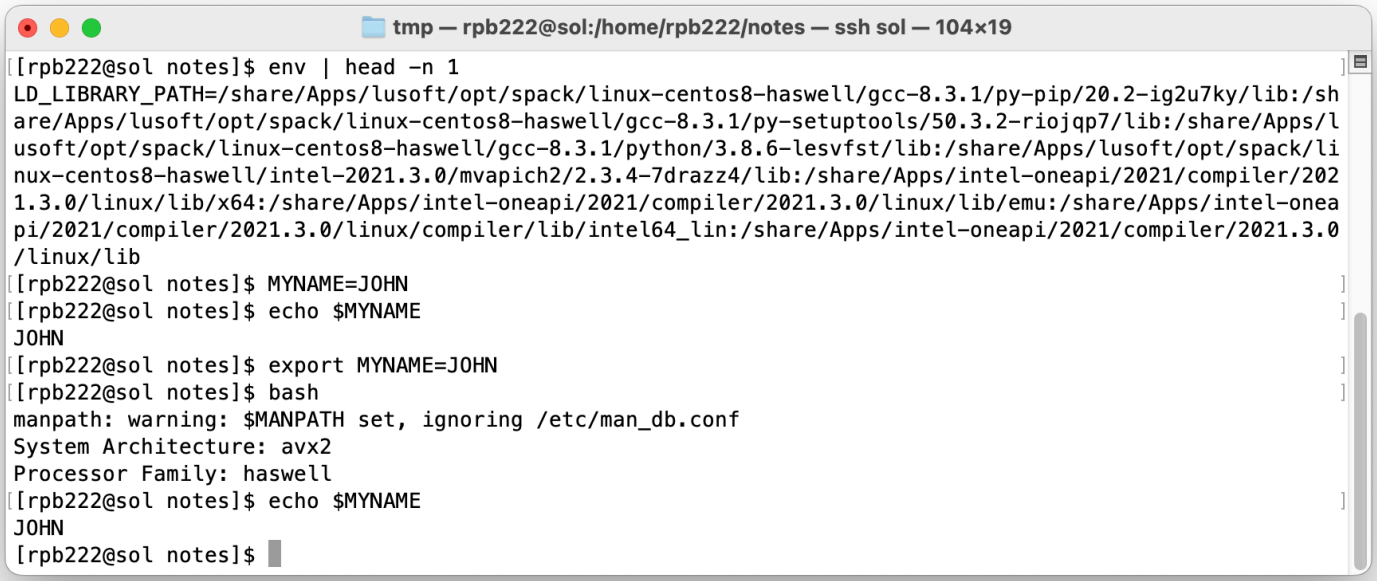


- To run commands in the background, use the ampersand (`&`).
- Use `jobs` to list the index and PIDs for background processes in this shell.
- Resume a job in the foreground by index with `fg <N>`
- Resume a job in the background by index with `bg <N>`
- Terminate a job by index with `kill %<N>` or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

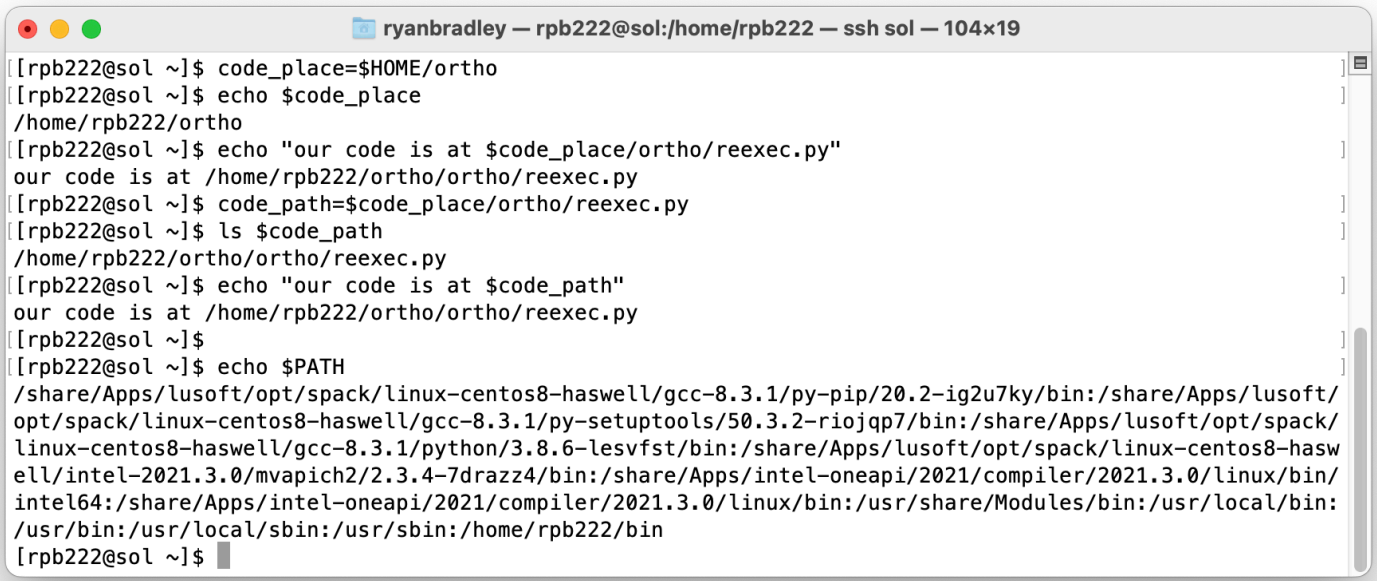
The Environment is a state



- 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.

LEHIGH UNIVERSITY

Custom variables



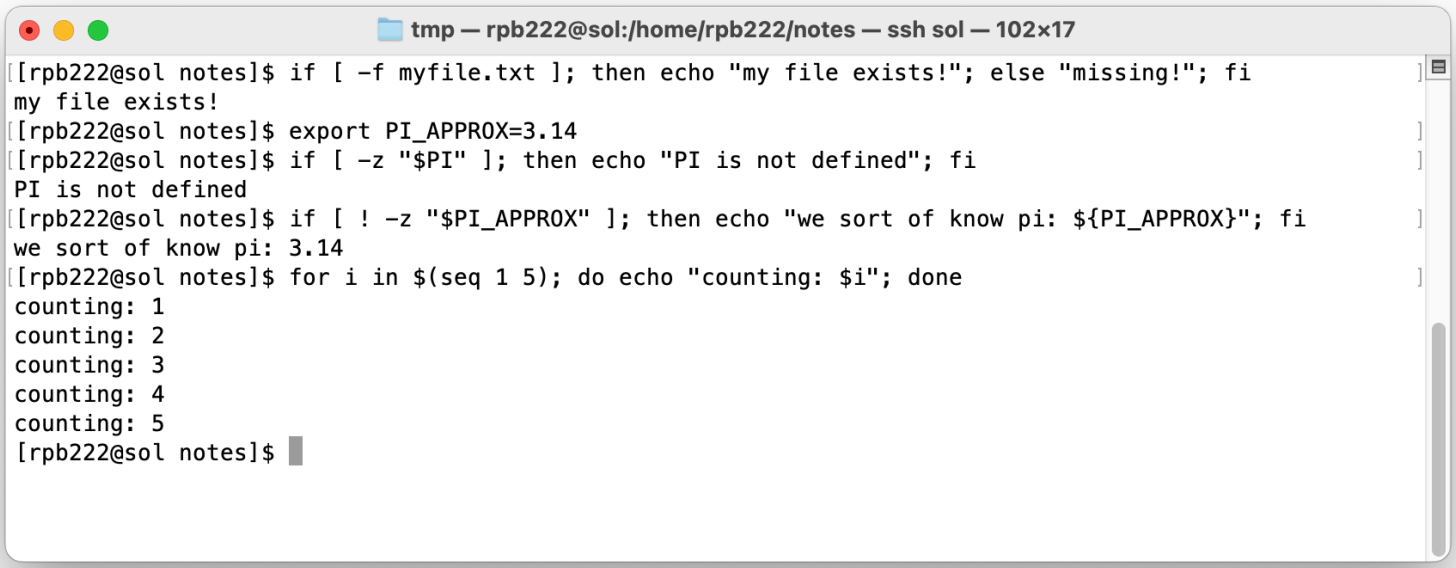
- Custom variables can be defined and reused.
- You can echo variables to standard output.
- You can compose new variables from existing ones.
- You can use variables as arguments.
- Many variables with specific names are used by compilers and the operating system.

LEHIGH UNIVERSITY

Startup

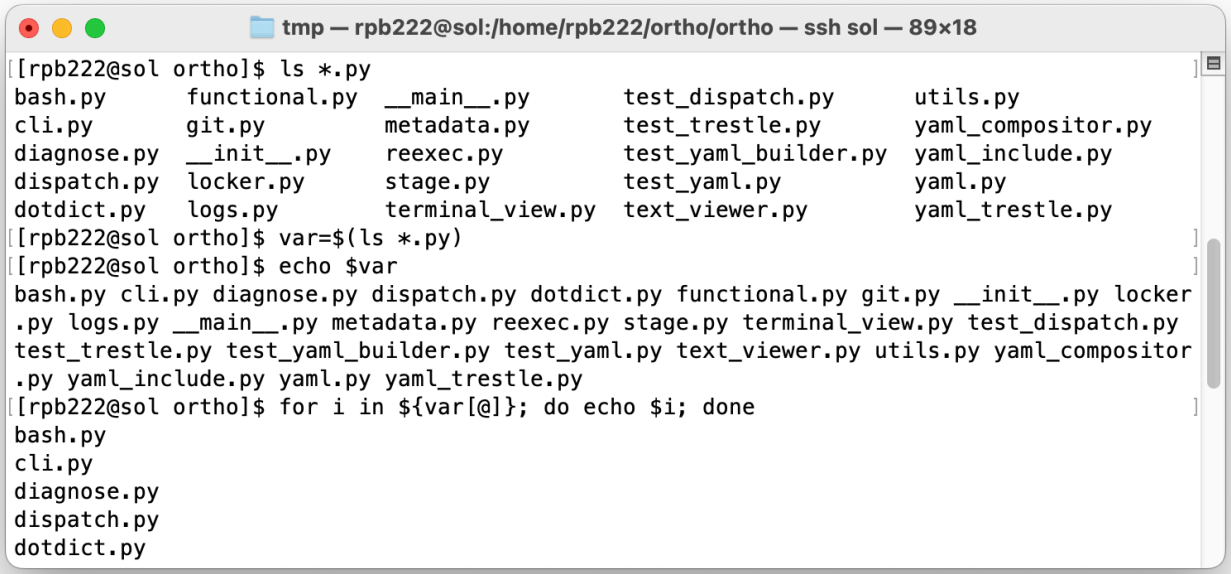
- Whenever you log into a ⋆NIX computer, shell scripts are automatically loaded.
- In BASH, we load:
  - `/etc/profile` (login terminal only)
  - `/etc/bashrc`
  - `$HOME/.bash_profile` (login terminal only)
  - `$HOME/.bashrc`
- Use the profile for user-facing tools, for example `alias` .
- Anything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

Loops and Conditionals



BASH is a fully-featured programming language

Arrays



- The BASH shell has many ways to work with lists.
- But they can be a bit more work than a traditional programming langague (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
```

LEHIGH UNIVERSITY

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

LEHIGH UNIVERSITY

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` .