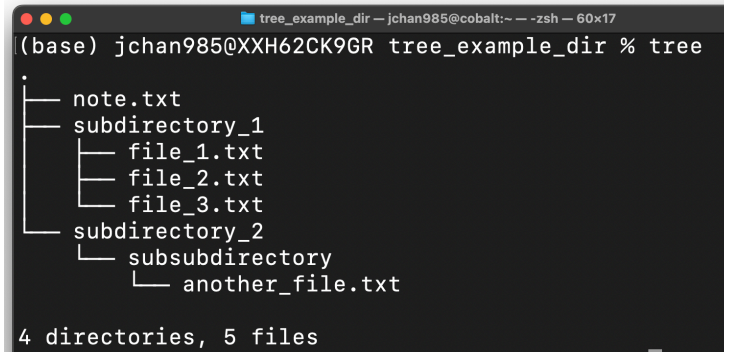
# CMOR 420/520
# Computational Science
# The Linux terminal

# Navigating the Linux terminal

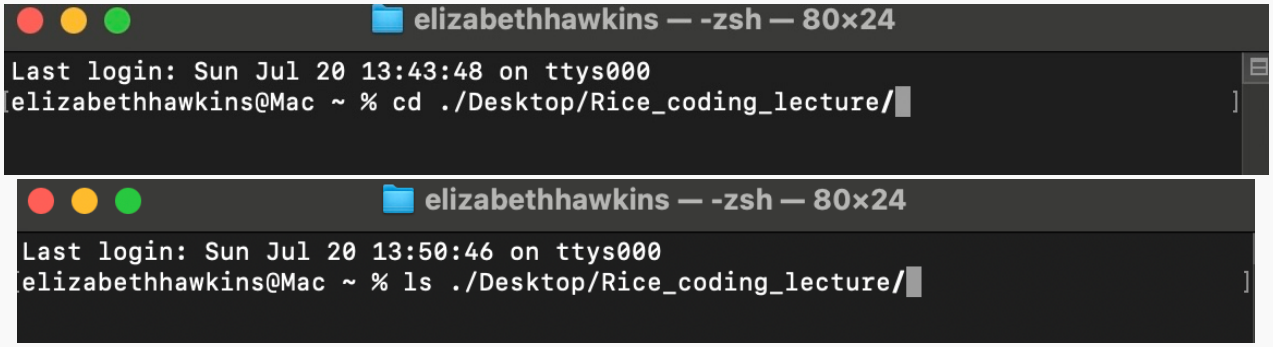
- Codes have a file system structure, can navigate with basic commands
  - pwd (show current directory name)
  - cd (change directory)




- Other useful commands:
  - ls (list files in current directory). Can add options (e.g., “ls -l” lists longer format file information)
  - Use “..” to refer to one directory up, “.” to refer to current directory, “~” refers to the *home* directory, …

# More on directory

- Address or path of a file or directory
- You can give commands the subdirectory or file you want by telling it the path: ./Directory-1/Directory-2/Directory-3



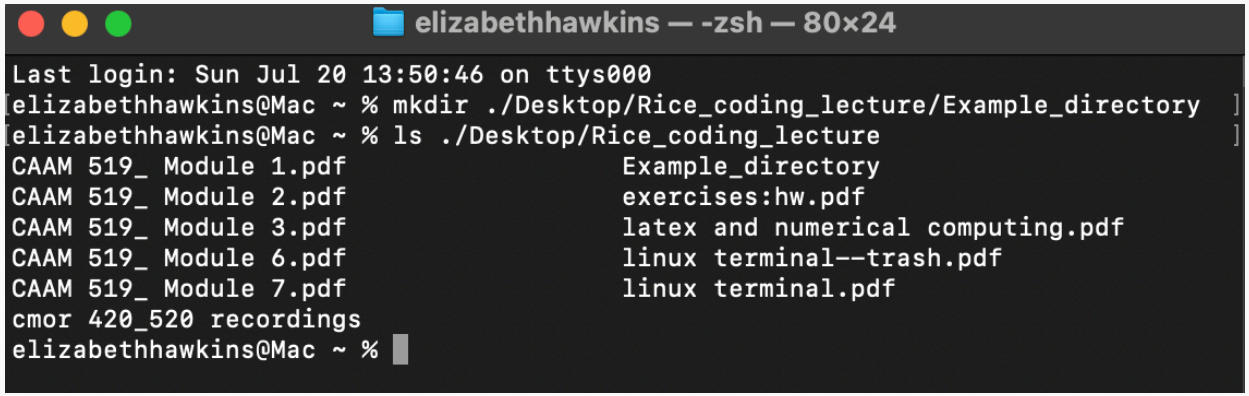

- You dont have to use the cd command multiple times!

# File management

- touch (create file), mkdir (create directory),
- mv (move file), cp (copy file). These commands can also be used to rename files.
- rm (remove file). Be very careful with this; no easy way to undo deletion actions in Linux!
- rmdir (remove directory). You can only remove empty directories.

# More on File Management

- Rememeber you dont have to use cd to enter into a directory in order to use a command! You just need the path to the file/directory you want.



# Exercise

Take a moment to familiarize yourself with your computer directory:
Use cd and ls to move through the directory

1. Make a file called 'Exercise_1'
2. Inside your 'Documents' directory, make a directory called 'CMOR'
3. Move the 'Exercise_1' file to the 'CMOR' directory

pwd : show name of current directory

cd : change directory

ls : show current directory contents

touch : create file

mkdir : make a directory

rmdir : remove a directory

mv : move a file

cp : copy a file

cat : display file contents

echo : insert text into file

wc : retuns word count of a file

# Terminal hotkeys, tips, and tricks

- Hotkeys and shortcuts make things a lot faster
  - Ctrl-a (home), Ctrl-e (go to end of line)
  - Ctrl-d (delete character), Ctrl-l (clear screen)
  - Ctrl-k (cut everything after terminal “cursor”), Ctrl-y (paste)
  - Ctrl-c (interrupt current command), Ctrl-g (leave history searching mode without running the command)
  - Ctrl-r (search backwards in the command history), Ctrl-s (search forwards)
- Autocomplete any word by pressing “tab” one or more times
- Use “man” to see what a command does (e.g., “man pwd”). Press “q” to quit.

# Wildcards (or “globbing”)

- Linux is great for automation; it’s harder in Linux to do simple things, but easier to do very complex things
- In Linux, * is a special wildcard character
  - Suppose I want to remove 3 files via “rm file_1.dat file_2.dat file_3.dat”. I could also do “rm file_*.dat”
  - How would you list all .txt files in a directory?
- Be careful; “rm *” will remove everything that it’s allowed to remove in the current directory.

# File permissions

- Can view file permissions using “ls -l” (list in long format)

```
(base) jchan985@XXH62CK9GR tree_example_dir % ls -l
total 8
-rw-r--r--  1 jchan985  staff   18 Aug 21 11:40 filename.txt
drwxr-xr-x  5 jchan985  staff  160 Aug 21 11:20 subdirectory_1
drwxr-xr-x  3 jchan985  staff   96 Aug 21 11:19 subdirectory_2
-rw-r--r--  1 jchan985  staff    0 Aug 21 11:41 wc
-rw-r--r--  1 jchan985  staff    0 Aug 21 11:41 wd
```

- “rwx” letters refer to **r**ead, **w**rite, and e**x**ecute permissions
- “drwxr-xr-x” means the **d**irectory is **r**eadable, **w**riteable, and e**x**ecutable by the *user*, and e**x**ecutable/**w**riteable by other user categories (the “*user group*” and “*others*”).

# Changing file permissions

- Change a file's permissions using "chmod"
  - "chmod u-r file.txt" removes (-) the read (r) permissions for the current user (u).
  - "chmod a+w file.txt" adds (+) write (w) permissions for all users (a).
- Often used to allow files to be executable (runnable)
  - e.g., "chmod u+x script.sh"
- Can override some restrictions, e.g., "sudo rm file.txt"

# Exercise

Open a terminal and run the ls -l command to view the contents

- Go to the 'CMOR' directory
- Remove the write (w) permission of 'Exercise_1'
- Check you have done this with ls -l
- Give the write (w) permission back to 'Exercise_1'

# How do we actually edit a file?

- If you have a native terminal (Mac or Linux) you can use a text editor (e.g., VSCode). Not sure about Windows.
- If you are using ssh, you usually need a terminal-based editor like Nano, Emacs, or Vim
  - If you open a text editor, you can hide it using Ctrl-z (minimize) and reveal it again using “fg” (“foreground”)
  - I use vim, I am not familiar with Nano, Emacs

# VIM

- vim filename.txt to open
- has a 'normal mode', 'insert mode', and 'visual mode'
  - normal mode allows for entering commands
  - insert mode allows for direct manipulation of content
  - visual mode allows for selecting blocks of text
- i – Enter insert mode at the character preceding the cursor
- a – Enter insert mode at the character following the cursor
- escape – leave insert mode
- v – Enter visual mode

# VIM

:w – Save the file (in normal mode)

:q – Exit the file (in normal mode)

:q! – Force exit the file, deleting unsaved changes (in normal mode)

dd – delete current line

p – paste clipboard

k – move up a line

j – move down a line

h – move left a character

l – more right a character

$ – jump to end of line

^ – jump to beginning of line

# Exercise

- Open ’Exercise.txt’ from the last class using vim
- Add any text to it, then exit and save

# ssh and scp

- Secure shell (SSH) protocol allows users to access another computer remotely.
- Secure copy (scp) allows you to copy files remotely
- All of you should have access to CLEAR through net_id@ssh.clear.rice.edu.

# Shell scripting

- Can run a series of Linux commands inside a “shell script”
- Change a file’s permissions using “chmod”
  - “chmod u-r file.txt” removes (-) the read (r) permissions for the user (u)
  - “chmod a+w file.txt” adds (+) write (w) permissions for all users (a)
- Often used to allow files to be executable (runnable)
  - e.g., “chmod u+x script.sh”

# Shell scripting, cont.

- Can store output as variables, e.g., dir=$(pwd).
- Can refer to variables, e.g., echo “Directory is $dir” (notice there is no parentheses here)
- Note that shell scripts are sensitive to tabs/spaces.
- Shell scripts start with a “shebang”: “#!/bin/bash” or “#!/usr/bin/bash” (depending on where “bash” is installed; to see where it is installed, run “which bash”).
  - “bash” and “zsh” are different shells with different features.

# Shell scripting examples

- Advanced shell scripting includes for/while loops, if statements, and even arrays.
- Demos:
  - first_shell_script.sh
  - arg_shell_script.sh

# Exercise

- Make a shell script called 'shell_exercise.sh' that:
  - Creates a file called 'shell_exercise.txt'
  - Moves the file to the 'CMOR' directory
  - Opens the file using vim

# Other useful information

- A file starting with a “.” is a hidden file. You can see these with the “-a” option to “ls”
- Run a command in the background using “&”, e.g., “command_name &” (can “foreground” the file via “fg”).
- Other useful commands: top (see running processes), kill (kill a process listed by “top”), and du (see disk usage).
- The shell records the history of commands executed

# Input, output, and redirect

- Linux commands have inputs and outputs
- Can use “>” to redirect an output to a file
  - “>>” to append the output to end of an existing file
  - Example: echo “Hello hello” > filename.txt
  - Example: pwd > filename.txt; ls >> filename.txt
- Can use “<“ to send file as input to argument
  - Example: what does wc < filename.txt do?

# Piping / chaining commands (and searching)

- Can chain commands together using the pipe “|”
  - command_1 | command_2 | .... | command_N
- Example: searching for text matches via “grep”
  - “grep ‘string-to-search-for’ text_to_search”
  - What would “cat ./*.sh | grep “fname” ” do?
  - Lots of useful flags: “-v” (print lines that *don’t* match), “-C n” (print “n” lines before/after a match), etc.

# Environment variables

- The Linux terminal utilizes “environment variables”
  - Consider all the programs we’ve run; how does Linux know where these are located?
  - Can run “echo $PATH” to see the directories that Linux searches when looking for a command
    - “$PATH” interpolates the variable PATH (e.g., substitutes the value of the variable into the
  - To add new directories to PATH, can run “export PATH=$PATH/new_directory_to_add”

# Package management (installation / update)

- Most Linux systems come with the package manager “apt” to help you easily install command software.
  - Example: “sudo apt install cowsay” (sudo: run as “root”)
  - On Mac, there is no “apt” package manager, but you can install the Homebrew package manager via https://brew.sh/
- Package managers keep a list of packages (a registry or manifest) that tracks new versions. Can update this manifest via “sudo apt update”.