Lecture 1

Tuesday, September 8th, 2020

Unix

Last Time

- Course introduction / policies
- Unix and Linux terminology
- The `ls` command

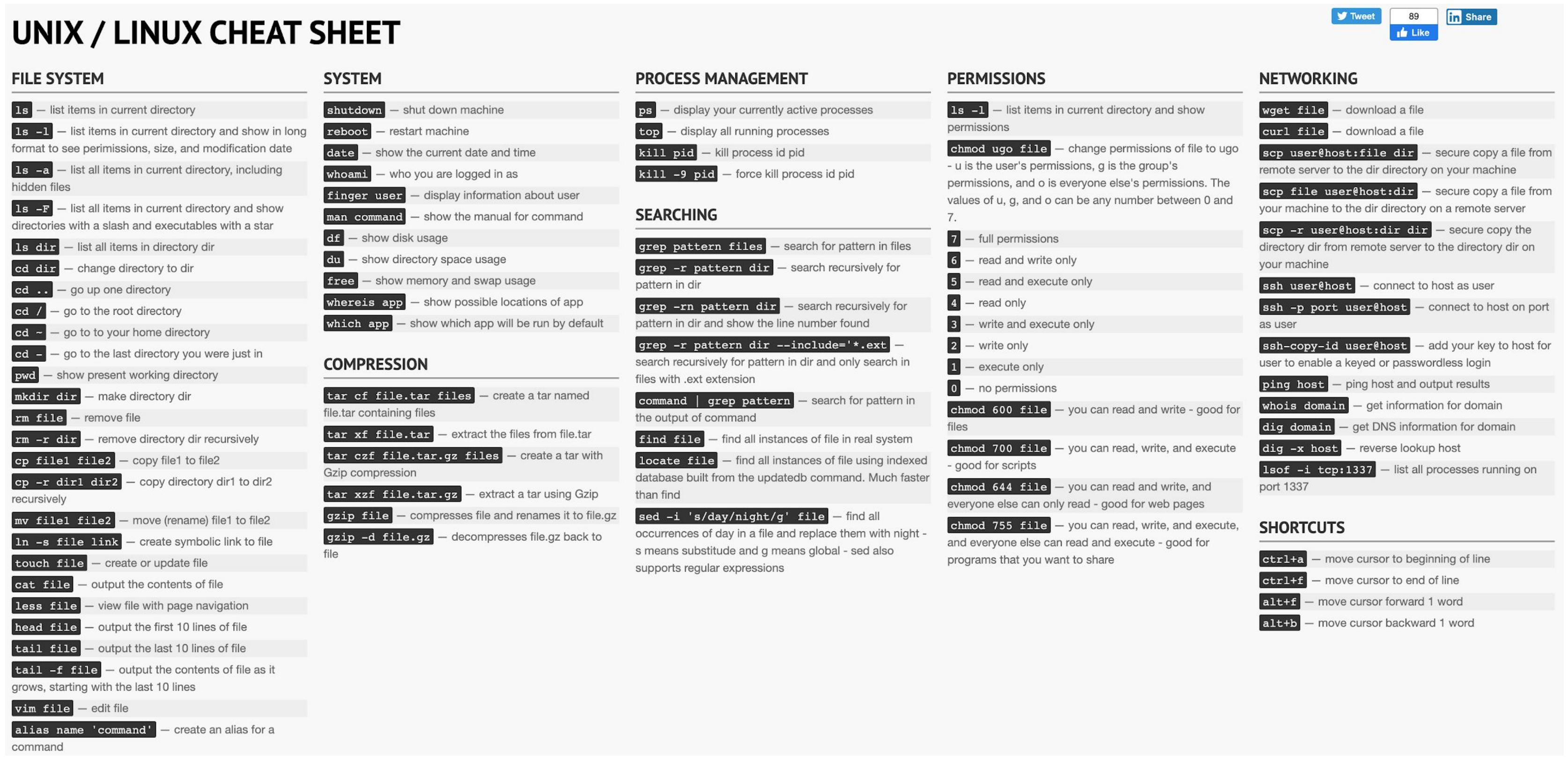
Today

- More on Unix / Linux
  - Unix commands
  - Interacting with the shell
  - File attributes
  - Text editors

After this lecture, you will know more about working with Unix and how to edit files.

Unix Commands

Basic Unix Commands



UNIX / LINUX CHEAT SHEET

FILE SYSTEM

- `ls` — list items in current directory
- `ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date
- `ls -a` — list all items in current directory, including hidden files
- `ls -F` — list all items in current directory and show directories with a slash and executables with a star
- `ls dir` — list all items in directory dir
- `cd dir` — change directory to dir
- `cd ..` — go up one directory
- `cd /` — go to the root directory
- `cd ~` — go to to your home directory
- `cd -` — go to the last directory you were just in
- `pwd` — show present working directory
- `mkdir dir` — make directory dir
- `rm file` — remove file
- `rm -r dir` — remove directory dir recursively
- `cp file1 file2` — copy file1 to file2
- `cp -r dir1 dir2` — copy directory dir1 to dir2 recursively
- `mv file1 file2` — move (rename) file1 to file2
- `ln -s file link` — create symbolic link to file
- `touch file` — create or update file
- `cat file` — output the contents of file
- `less file` — view file with page navigation
- `head file` — output the first 10 lines of file
- `tail file` — output the last 10 lines of file
- `tail -f file` — output the contents of file as it grows, starting with the last 10 lines
- `vim file` — edit file
- `alias name 'command'` — create an alias for a command

SYSTEM

- `shutdown` — shut down machine
- `reboot` — restart machine
- `date` — show the current date and time
- `whoami` — who you are logged in as
- `finger user` — display information about user
- `man command` — show the manual for command
- `df` — show disk usage
- `du` — show directory space usage
- `free` — show memory and swap usage
- `whereis app` — show possible locations of app
- `which app` — show which app will be run by default

COMPRESSION

- `tar cf file.tar files` — create a tar named file.tar containing files
- `tar xf file.tar` — extract the files from file.tar
- `tar czf file.tar.gz files` — create a tar with Gzip compression
- `tar xzf file.tar.gz` — extract a tar using Gzip
- `gzip file` — compresses file and renames it to file.gz
- `gzip -d file.gz` — decompresses file.gz back to file

PROCESS MANAGEMENT

- `ps` — display your currently active processes
- `top` — display all running processes
- `kill pid` — kill process id pid
- `kill -9 pid` — force kill process id pid

SEARCHING

- `grep pattern files` — search for pattern in files
- `grep -r pattern dir` — search recursively for pattern in dir
- `grep -rn pattern dir` — search recursively for pattern in dir and show the line number found
- `grep -r pattern dir --include='*.ext` — search recursively for pattern in dir and only search in files with .ext extension
- `command | grep pattern` — search for pattern in the output of command
- `find file` — find all instances of file in real system
- `locate file` — find all instances of file using indexed database built from the updatedb command. Much faster than find
- `sed -i 's/day/night/g' file` — find all occurrences of day in a file and replace them with night - s means substitude and g means global - sed also supports regular expressions

PERMISSIONS

- `ls -l` — list items in current directory and show permissions
- `chmod ugo file` — change permissions of file to ugo - u is the user's permissions, g is the group's permissions, and o is everyone else's permissions. The values of u, g, and o can be any number between 0 and 7.
- `7` — full permissions
- `6` — read and write only
- `5` — read and execute only
- `4` — read only
- `3` — write and execute only
- `2` — write only
- `1` — execute only
- `0` — no permissions
- `chmod 600 file` — you can read and write - good for files
- `chmod 700 file` — you can read, write, and execute - good for scripts
- `chmod 644 file` — you can read and write, and everyone else can only read - good for web pages
- `chmod 755 file` — you can read, write, and execute, and everyone else can read and execute - good for programs that you want to share

NETWORKING

- `wget file` — download a file
- `curl file` — download a file
- `scp user@host:file dir` — secure copy a file from remote server to the dir directory on your machine
- `scp file user@host:dir` — secure copy a file from your machine to the dir directory on a remote server
- `scp -r user@host:dir dir` — secure copy the directory dir from remote server to the directory dir on your machine
- `ssh user@host` — connect to host as user
- `ssh -p port user@host` — connect to host on port as user
- `ssh-copy-id user@host` — add your key to host for user to enable a keyed or passwordless login
- `ping host` — ping host and output results
- `whois domain` — get information for domain
- `dig domain` — get DNS information for domain
- `dig -x host` — reverse lookup host
- `lsof -i tcp:1337` — list all processes running on port 1337

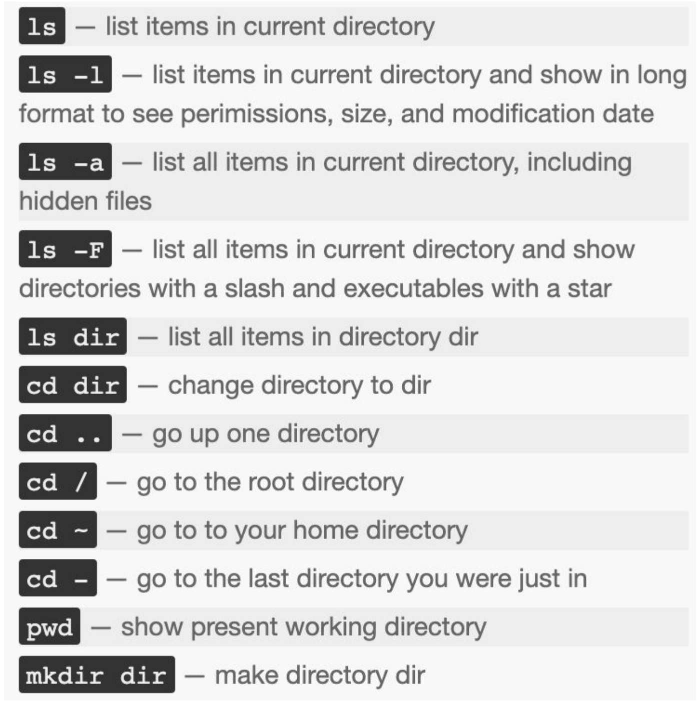
SHORTCUTS

- `ctrl+a` — move cursor to beginning of line
- `ctrl+f` — move cursor to end of line
- `alt+f` — move cursor forward 1 word
- `alt+b` — move cursor backward 1 word

http://cheatsheetworld.com/programming/unix-linux-cheat-sheet/

Absolutely Essential Commands

These commands should be at your fingertips at all times



`ls` — list items in current directory

`ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date

`ls -a` — list all items in current directory, including hidden files

`ls -F` — list all items in current directory and show directories with a slash and executables with a star

`ls dir` — list all items in directory dir

`cd dir` — change directory to dir

`cd ..` — go up one directory

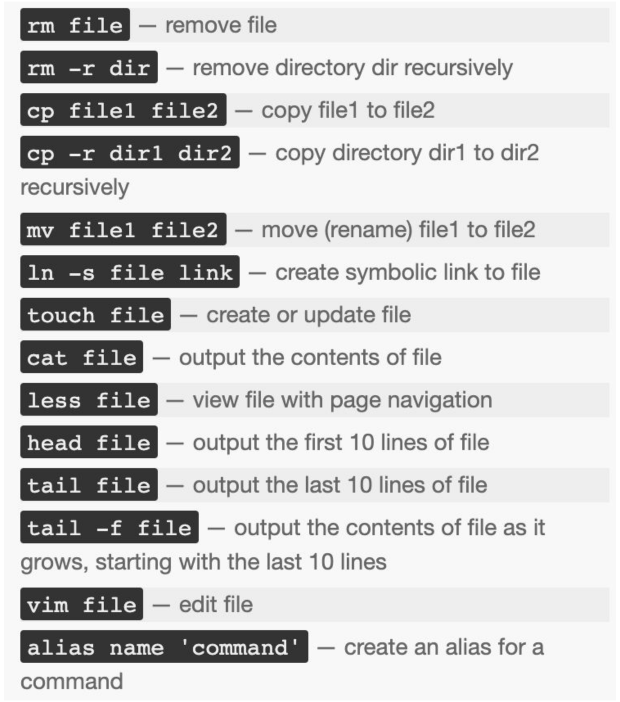
`cd /` — go to the root directory

`cd ~` — go to to your home directory

`cd -` — go to the last directory you were just in

`pwd` — show present working directory

`mkdir dir` — make directory dir



`rm file` — remove file

`rm -r dir` — remove directory dir recursively

`cp file1 file2` — copy file1 to file2

`cp -r dir1 dir2` — copy directory dir1 to dir2 recursively

`mv file1 file2` — move (rename) file1 to file2

`ln -s file link` — create symbolic link to file

`touch file` — create or update file

`cat file` — output the contents of file

`less file` — view file with page navigation

`head file` — output the first 10 lines of file

`tail file` — output the last 10 lines of file

`tail -f file` — output the contents of file as it grows, starting with the last 10 lines

`vim file` — edit file

`alias name 'command'` — create an alias for a command

`man` and More Information

- **man pages** (manual pages) provide extensive documentation
- The Unix command to display a manual page is `man`
- Man pages are split into 8 numbered sections
    - 1. General commands
    - 2. System calls
    - 3. `C` library functions
    - 4. Special files (usually devices found in `/dev`)
    - 5. File formats and conventions
    - 6. Games
    - 7. Miscellaneous
    - 8. Sys admin commands and daemons
- You can request pages from specific sections, e.g.
    - `man 3 printf` (shows man page for `C` library function)

Breakout Room

- Figure out who is at the most northern latitude in your group.
- Choose one of the Unix commands from the cheatsheet. Read about it using `man`. Try it out a bit as a group. Make sure you can all provide a short summary of what it does. What is one interesting option that this command provides?

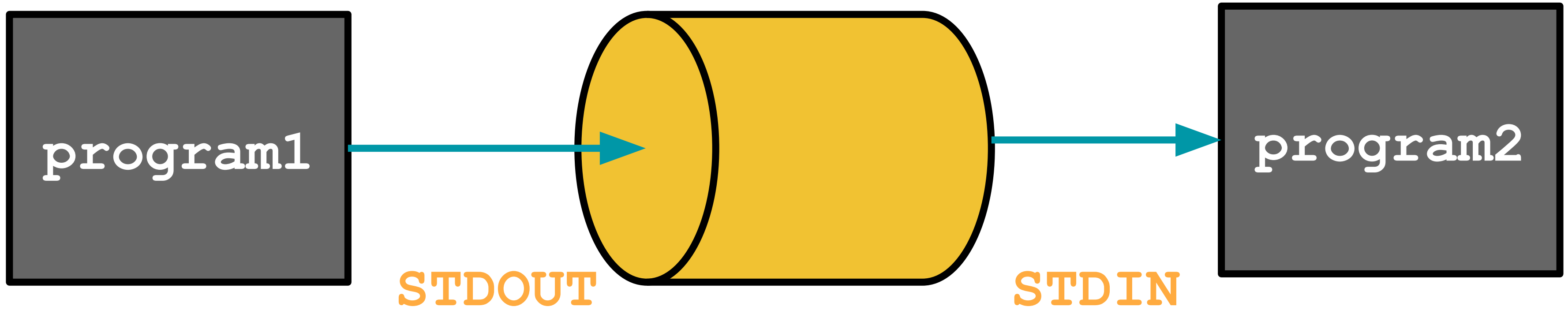
Interacting with the Shell

Running a Unix Program

- Type in the name of a program and some command line options
- The shell reads this line, finds the program, and runs it feeding it the options you specified
- The shell establishes three I/O streams:
  - 1. Standard input
  - 2. Standard output
  - 3. Standard error
- File descriptors associated with each stream:
  - 0 = STDIN (standard input)
  - 1 = STDOUT (standard output)
  - 2 = STDERR (standard error)

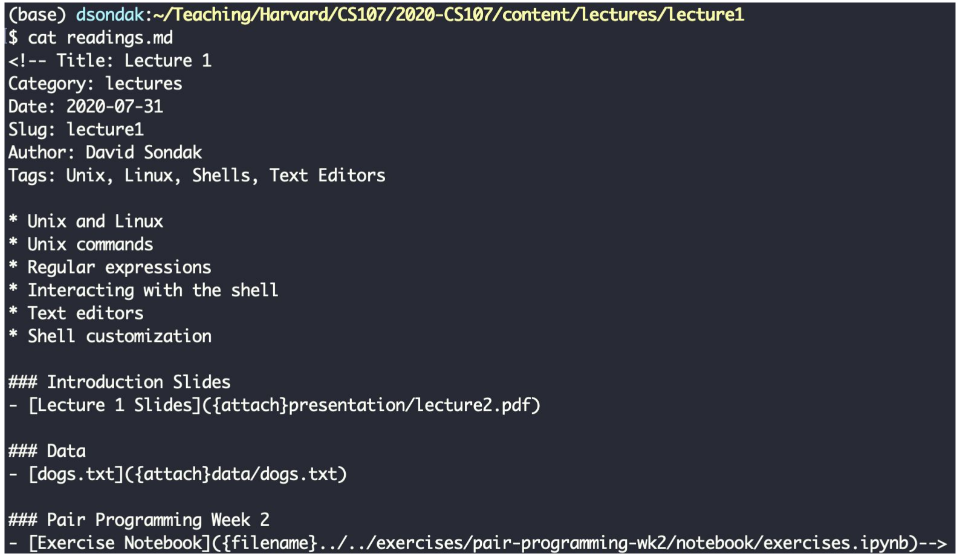
Unix Pipes

- A pipe is a holder for a stream of data
- A Unix **pipeline** is a set of processes chained by their standard streams
    - The output of each process (`stdout`) feeds directly as input (`stdin`) to the next one
- Very useful for using multiple Unix commands together to perform a task

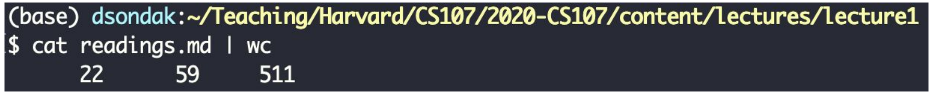


Building Commands

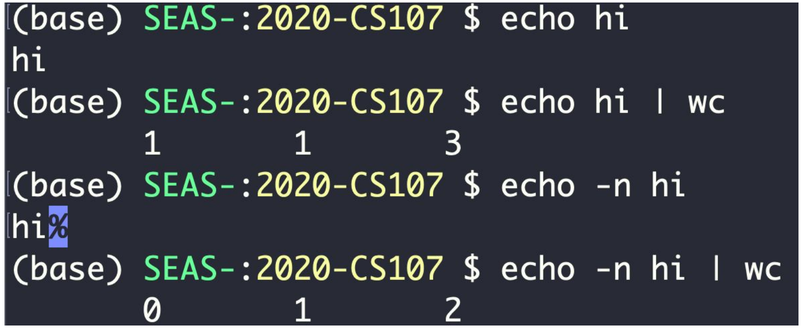
- More complicated commands can be built up by using one or more pipes
- The `|` character is used to *pipe* two commands together
- The shell does the rest for you!



```
(base) dsondak:~/Teaching/Harvard/CS107/2020-CS107/content/lectures/lecture1
$ cat readings.md
<!-- Title: Lecture 1
Category: lectures
Date: 2020-07-31
Slug: lecture1
Author: David Sondak
Tags: Unix, Linux, Shells, Text Editors

* Unix and Linux
* Unix commands
* Regular expressions
* Interacting with the shell
* Text editors
* Shell customization

### Introduction Slides
- [Lecture 1 Slides]({attach}presentation/lecture2.pdf)

### Data
- [dogs.txt]({attach}data/dogs.txt)

### Pair Programming Week 2
- [Exercise Notebook]({filename}../../exercises/pair-programming-wk2/notebook/exercises.ipynb)-->
```



```
(base) dsondak:~/Teaching/Harvard/CS107/2020-CS107/content/lectures/lecture1
$ cat readings.md | wc
      22      59     511
```

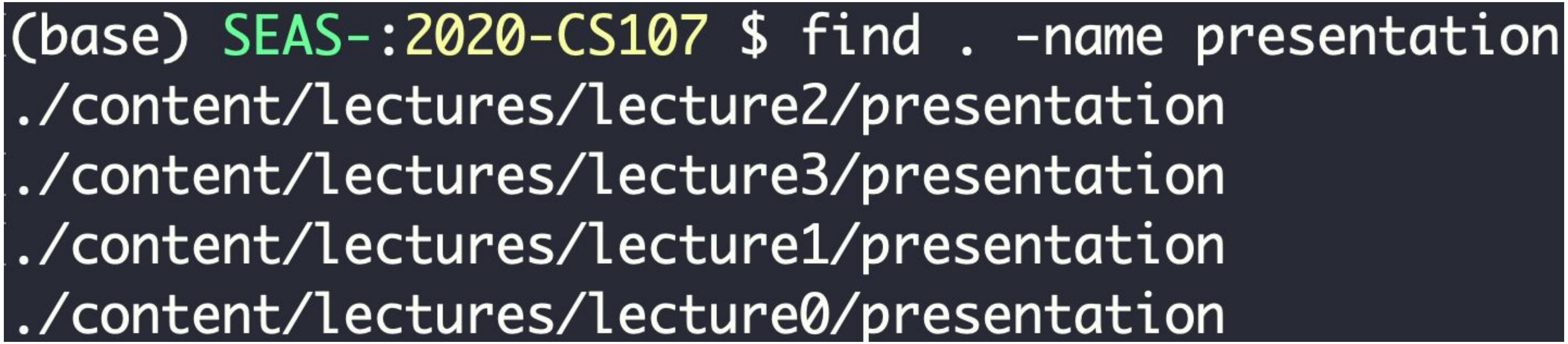


```
(base) SEAS-:2020-CS107 $ echo hi
hi
(base) SEAS-:2020-CS107 $ echo hi | wc
       1       1       3
(base) SEAS-:2020-CS107 $ echo -n hi
hi%
(base) SEAS-:2020-CS107 $ echo -n hi | wc
       0       1       2
```

- Note: `wc` prints the number of newlines, words, and bytes in a file

More Unix Commands: `find`

- `find` searches the filesystem for files whose name matches a specific pattern
- It can do much more than this and is one of the most useful commands in Unix
  - e.g. It can find files and then perform operations on them
- Example:



```
(base) SEAS-:2020-CS107 $ find . -name presentation
./content/lectures/lecture2/presentation
./content/lectures/lecture3/presentation
./content/lectures/lecture1/presentation
./content/lectures/lecture0/presentation
```

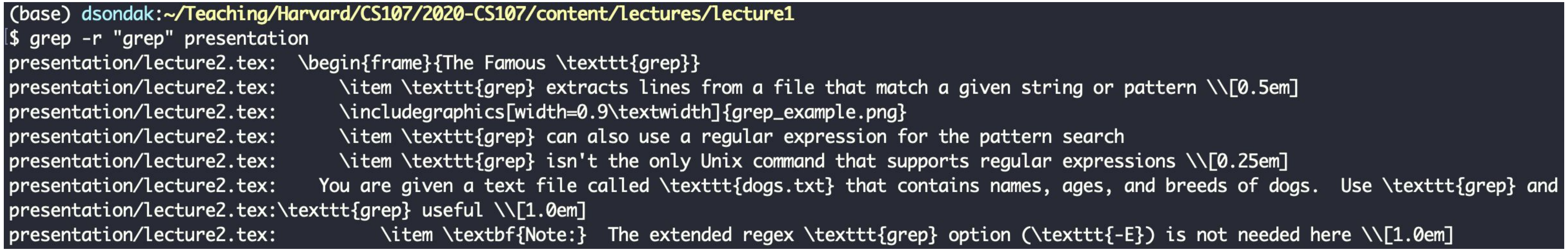
`find`

- `find` can also scan for certain file types:
  - Find directories with `find . -type d`
  - Find files with `find . -type f`
- The `exec` option can be used to make very powerful commands on files
  - `find . -type f -exec wc -l {} \;`
- What does this command do?

- Find files (`-type f`) in the current directory (`.`) and execute (`-exec`) the word count command (`wc`) on them with the line count option (`-l`).
  - The current file gets put into the `{}`
  - The `;` is used to terminate the command invoked by `-exec`
  - Need the `\` in front of `;` to tell the shell to interpret `;` correctly

The Famous `grep`

- *"Global regular expression print"*
- `grep` extracts lines from a file that match a given string or pattern
- `grep` can also use a *regular expression* for the pattern search



```
(base) dsondak:~/Teaching/Harvard/CS107/2020-CS107/content/lectures/lecture1
$ grep -r "grep" presentation
presentation/lecture2.tex:  \begin{frame}{The Famous \texttt{grep}}
presentation/lecture2.tex:      \item \texttt{grep} extracts lines from a file that match a given string or pattern \\[0.5em]
presentation/lecture2.tex:      \includegraphics[width=0.9\textwidth]{grep_example.png}
presentation/lecture2.tex:      \item \texttt{grep} can also use a regular expression for the pattern search
presentation/lecture2.tex:      \item \texttt{grep} isn't the only Unix command that supports regular expressions \\[0.25em]
presentation/lecture2.tex:    You are given a text file called \texttt{dogs.txt} that contains names, ages, and breeds of dogs.  Use \texttt{grep} and
presentation/lecture2.tex:\texttt{grep} useful \\[1.0em]
presentation/lecture2.tex:          \item \textbf{Note:}  The extended regex \texttt{grep} option (\texttt{-E}) is not needed here \\[1.0em]
```

- `grep` isn't the only Unix command that supports regular expressions
  - `sed`
  - `awk`
  - `perl`

Regular Expressions

- General search pattern characters
  - Any character
  - `.` matches any character except a newline
  - `*` matches zero or more occurrences of the single preceding character
  - `+` matches one or more of the proceeding character
  - `?` matches zero or one of the proceeding character
- More special characters
  - `()` are used to quantify a sequence of characters
  - `|` functions as an OR operator
  - `{}` are used to indicate ranges in the number of occurrences

More on Regular Expressions

- To match a special character, you should use the backslash `\`
  - To match a period do `\.`
  - `a\.b` matches `a.b` because `.` is special
- A *character class* (a.k.a. character set) can be used to match *only one* out of several characters
- Place the characters you want to match between square brackets `[]`
- A hyphen can be used to specify a range of characters
- A caret, `^`, after the opening square bracket will negate the class
  - The result is that the character will match any character that is **not** in the character class
- Examples
  - `[abc]` matches a single `a`, `b`, `c`
  - `[0-9]` matches a single digit between `0` and `9`
  - `[^A-Za-z]` matches a single character as long as it's not a letter

Regular Expressions Continued

- Some shorthand character classes are available for convenience
    - `\d` a digit, e.g. `[0-9]`
    - `\D` a non-digit, e.g. `[^0-9]`
    - `\w` a word character, matches letters and digits
    - `\W` a non-word character
    - `\s` a whitespace character
    - `\S` a non-whitespace character
- Some shorthand classes are available for matching boundaries
    - `^` the beginning of a line
    - `$` the end of a line
    - `\b` a word boundary
    - `\B` a non-word boundary
- Some references
    - https://regexone.com/
    - Mastering Regular Expressions, 3rd Edition [Book]

File Attributes

File Attributes

Every file has a specific list of attributes:

- Access times
    - when the file was created
    - when the file was last changed
    - when the file was last read
- Size
- Owners
    - user (remember `UID`?)
    - group (remember `GID`?)
- Permissions

Quick Examples

For example, time attributes access with `ls`

- `ls -l` shows when the file was last changed
- `ls -lc` shows when the file was created
- `ls -lu` shows when the file was last accessed

File Permissions

- Each file has a set of permissions that control who can access the file
- There are three different types of permissions:
  - read, abbreviated `r`
  - write, abbreviated `w`
  - execute, abbreviated `x`
- In Unix, there are permission levels associated with three types of people that might access a file:
  - owner (you)
  - group (a group of other users that you set up)
  - world (anyone else browsing around on the file system)

File Permissions Display Format

`-` `rwx` `rwx` `rwx`

Owner Group Others

- The first entry specifies the type of file:
  - `-` is a plain file
  - `d` is a directory
  - `c` is a character device
  - `b` is a block device
  - `l` is a symbolic link

- Meaning for files
  - `r` - allowed to read
  - `w` - allowed to write
  - `x` - allowed to execute

- Meaning for directories
  - `r` - allowed to see the names of files
  - `w` - allowed to add and remove files
  - `x` - allowed to enter the directory

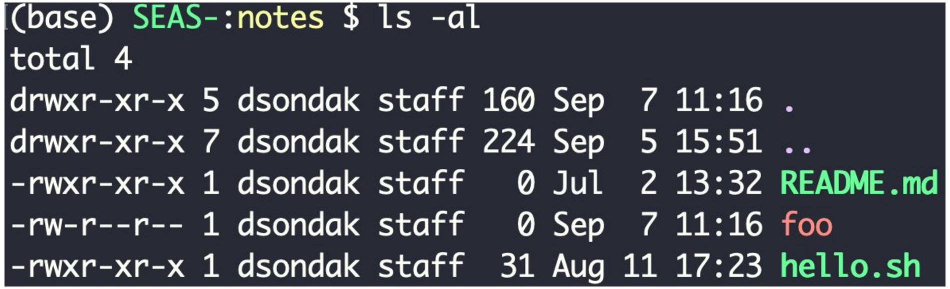
Changing File Permissions

- The `chmod` command changes the permissions associated with a file or directory
- Basic syntax: `chmod <mode> <file>`
- The `<mode>` can be specified in two ways:
    - Symbolic representation
    - Octal number
- It's up to you which method you use
- Multiple symbolic operations can be given, separated by commas

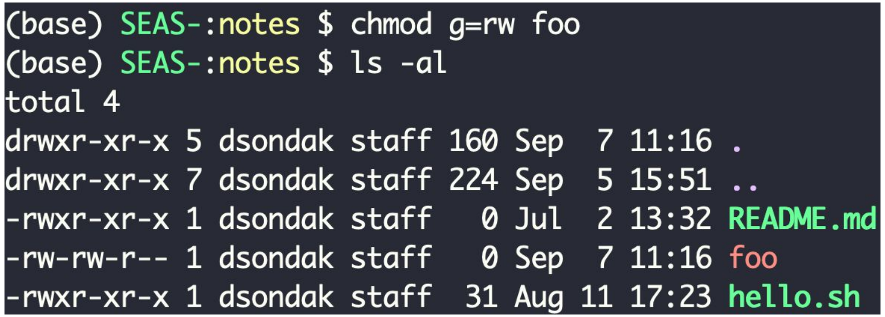
Symbolic Representation

- Symbolic representation has the following form
  - `[ugoa] [+-=] [rwxX]`
- `u=user` `g=group` `o=other` `a=all`
- `+` add permission `-` remove permission `=` set permission
- `r=read` `w=write` `x=execute`
- `X` sets to execute only if the file is a directory or already has execute permission
  - Very useful when using recursively

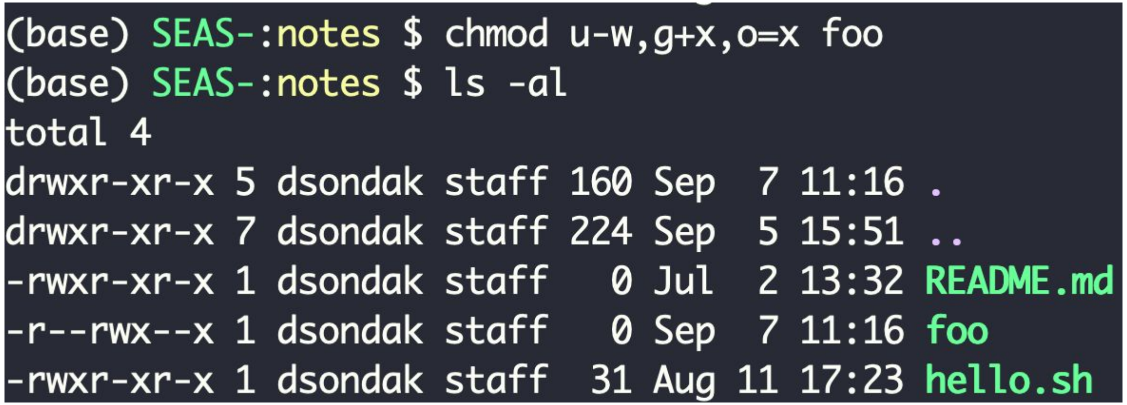
Symbolic Representation Examples



```
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 5 dsondak staff 160 Sep  7 11:16 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-rw-r--r-- 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```



```
(base) SEAS-:notes $ chmod g=rw foo
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 5 dsondak staff 160 Sep  7 11:16 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-rw-rw-r-- 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```



```
(base) SEAS-:notes $ chmod u-w,g+x,o=x foo
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 5 dsondak staff 160 Sep  7 11:16 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-r--rwx--x 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```

Octal Representation

- Octal mode uses a single-argument string which describes the permissions for a file (3 digits)
- Each digit is a code for each of the three permission levels
- Permissions are set according to the following numbers:
    - `read=4` `write=2` `execute=1`
- Sum the individual permissions to get the desired combination

`0` = no permission at all

`1` = execute only

`2` = write only

`3` = write and execute (`1+2`)

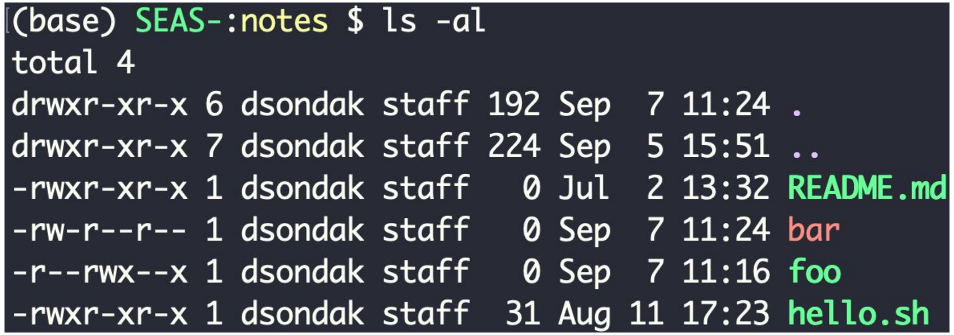
`4` = read only

`5` = read and execute (`4+1`)

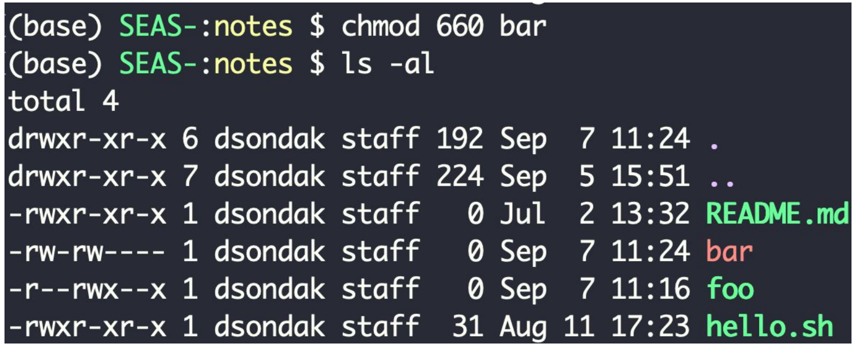
`6` = read and write (`4+2`)

`7` = read, write and execute (`4+2+1`)

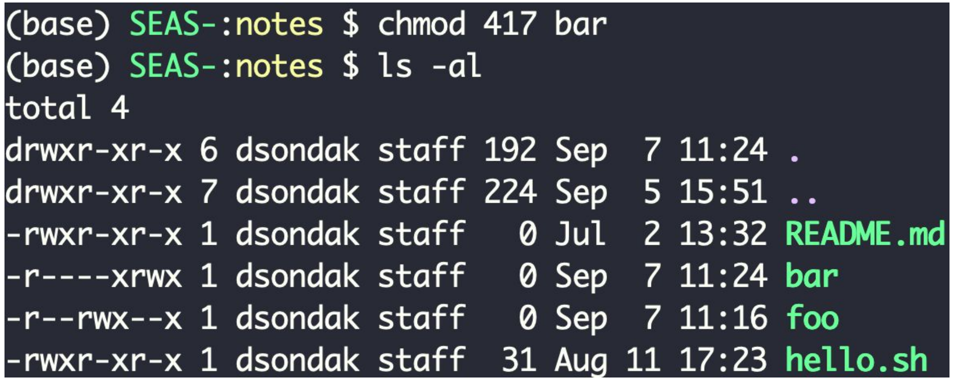
Octal Representation Examples



```
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 6 dsondak staff 192 Sep  7 11:24 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-rw-r--r-- 1 dsondak staff   0 Sep  7 11:24 bar
-r--rwx--x 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```



```
(base) SEAS-:notes $ chmod 660 bar
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 6 dsondak staff 192 Sep  7 11:24 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-rw-rw---- 1 dsondak staff   0 Sep  7 11:24 bar
-r--rwx--x 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```



```
(base) SEAS-:notes $ chmod 417 bar
(base) SEAS-:notes $ ls -al
total 4
drwxr-xr-x 6 dsondak staff 192 Sep  7 11:24 .
drwxr-xr-x 7 dsondak staff 224 Sep  5 15:51 ..
-rwxr-xr-x 1 dsondak staff   0 Jul  2 13:32 README.md
-r----xrwx 1 dsondak staff   0 Sep  7 11:24 bar
-r--rwx--x 1 dsondak staff   0 Sep  7 11:16 foo
-rwxr-xr-x 1 dsondak staff  31 Aug 11 17:23 hello.sh
```

Breakout Room

- Figure out who is in the most southern latitude.
- What does `chmod 777` do? Discuss some of the repercussions.

Text Editors

Text Editors

- We need to make use of available Unix text editors for programming and changing of various text files
- Two of the most popular and available text editors are vi and emacs
- You should familiarize yourself with at least one of the two
  - Editor Wars
- We will have very short introductions to each

A Brief Text Editor History

- `ed`: line mode editor
- `ex`: extended version of `ed`
- `vi`: full screen version of `ex`
- `vim`: Vi IMproved
- `emacs`: another *very* popular editor (but it's more than that…)

`ed`/`ex`/`vi` share lots of syntax, which can be found in `sed`/`awk` --- useful to know

`vi` overview

- The big thing to remember about `vi` is that it has two different modes of operation
  - 1.) Insert mode
  - 2.) Command mode
- The insert mode puts anything typed on the keyboard into the current file
- The command mode allows the entry of command to manipulate text
- Note that `vi` starts out in command mode by default

`vim` Quickstart Commands

- `vim <filename>`
- Press `i` to enable insert mode
- Type text (use arrow keys to move around)
- Press `Esc` to enable command mode
- Press `:w` (followed by `return`) to save the file
- Press `:q` (followed by `return`) to exit `vim`

This may feel strange at first, but you have to start somewhere. You’ll quickly learn to love it.

Useful `vim` commands

- :q! - exit without saving the document. Very handy for beginners!
- :wq - save and exit
- / <string> - search within the document for text.
    - n goes to the next result
- dd - delete the current line
- yy - copy the current line
- p - past the last cut/deleted line
- :1 - goto first line in the file
- :$ - goto last line in the file
- $ - end of current line
- ^ - beginning of line
- % - show matching brace, bracket, parentheses

Useful `vim` resources

Vim Cheat Sheet

Other Vim Cheat Sheets

vimtutor

VIM Adventures: Learn VIM while playing a game

A Note on IDEs

Many people use Interactive Development Environments (IDEs)

Examples include:

- Spyder
- Eclipse
- PyCharm

These can be very convenient and powerful, but they can also be rather bulky.

A lightweight text editor like `vim` (also `nano`, `atom`, etc) is quick and easy to use.

`vim` has the additional advantage of being available on almost any system by default.

Recap

Now you can:

- Learn even more Unix commands and really start using them
- Interact with the shell by searching for files and patterns
- Change file permissions and access
- Start editing files!