

# Missing Semester of CS Notes

Sean Wu

April 2, 2020

## Contents

<b>1</b>	<b>The Shell - Bash</b>	<b>2</b>
1.1	Paths . . . . .	2
1.2	Flags and Options . . . . .	3
1.3	File Permissions . . . . .	3
1.4	Deleting things . . . . .	4
1.5	Input and Output Streams . . . . .	4
1.6	Root User (Unix) . . . . .	4
1.7	Misc. Helpful Commands . . . . .	5
1.8	Executable and Unix Shebang . . . . .	6

# 1 The Shell - Bash

## 1.1 Paths

- Cmd line arguments separated by whitespace
- Use quotes " " or escape the space \

**environment variable:** variable set whenever shell starts (not every run of shell)

- ex. home dir, username, PATH variable
- Comments in Bash start with #

```
echo $PATH # all file paths that bash will search for programs
# OUTPUT: colon-separated list
```

- Whenever name of program (ex. `echo`) is typed, bash will search through this list in `PATH` , looking in each directory for the program matching the command

```
which echo # tells you where file for command is located (ex. echo)
```

**paths:** way to name location of file on computer

- Paths separated by forward slashes / for Unix and backslashes \ for Windows

/ root; top of file system

- On Unix, everything is under the root / namespace
  - i.e. all absolute paths start with /
- On Windows, there is one root for every partition
- ex. C:\, D:\
- i.e. separate file system path hierarchies for each drive

**absolute path:** fully determines location of file

**relative path:** path relative to your current working directory

. current directory

.. parent directory

~ home directory

- directory you were just in

## 1.2 Flags and Options

- Flags and options specified after the program name
- The short form is usually with single slashes -<char> and the long form is usually with double dashes --<word>
- ex. -v and --version tell you the version of the program
- ex. -h and --help give you a quick help guide for the program
- Running command with --help flag gives you the usage in the following format

```
usage: ls [OPTION] ... [FILE] ...  
# [] means optional  
# ... means 1 or more of the previous thing
```

**flag:** doesn't take a value (usually)

**option:** takes a value (usually)

## 1.3 File Permissions

- Get file permissions by running `ls -a`
- Permissions specified in 3 groups of 3 (r, w, x)
  1. 1st group of 3 permissions is for owner of file
  2. 2nd group of 3 permissions is for the group of people owning the file
  3. 3rd group of 3 permissions is for everyone else
- Note: if you have write access on a file but read access on a directory, you cannot directly delete a file (can only empty it)

**For files:**

- don't have that permission
- r** read access
- w** write access
- x** execute access

**For folders:**

- don't have that permission
- r** can see files inside directory
- w** can rename, create, remove files

`x` can search this directory (i.e. enter directory with `cd`)

`chmod` : command to change file modes or Access Control Lists (i.e. change permissions)

## 1.4 Deleting things

`rm` : removes a file

- By default, `rm` is **not** recursive on Unix (i.e. cannot remove a directory)
- Add a `-r` (recursive) flag to delete a directory
- Recursive delete removes everything under the path you give it

`rmdir` : deletes a directory only if it is empty (a safe delete)

`cmd L` : clears terminal output to previous mark

`cmd K` : clears terminal to start

## 1.5 Input and Output Streams

- Each program has 2 primary streams
  1. Input stream: terminal by default
  2. Output stream: terminal by default

`<` : rewire input of previous program to be the contents of this file on the right

`>` : rewire output of previous program into this file

`>>` : appends to the end of a file instead of overwriting

```
echo hello > hello.txt # writes string "hello" into file hello.txt
```

`|` : a **pipe**; takes the output of program on left and makes it the input of the program on the right. **Input program does not know about output program and vice versa** . The programs just read and write to those spots.

## 1.6 Root User (Unix)

- Acts like admin user on Windows

- Has user id 0
- Has all permissions (Superuser)

`sudo` : does the following command as superuser (root user)

**kernel:** core of computer

`sysfs` : file system for kernel parameters of computer

- Need to be admin to change kernel params of a computer
- Note: if using `sudo` with pipes and redirects, `sudo` only applies to one portion (because input and output programs don't know about each other)

\$ indicates that you are **not** running as root

# indicates that you are running as root

```
sudo echo 500 > brightness
# does not work because brightness doesn't know about sudo
```

`sudo su` gives you a shell as superuser (shell runs as root now)

`exit` allows you to exit out of superuser shell mode

## 1.7 Misc. Helpful Commands

`man` gives you the manual pages for a program

`tail` gives you the last n lines of a file

```
tail -n5 # gives you the last 5 lines of a file
```

`tee` writes to output and to terminal output

```
echo 1000 | sudo tee brightness # changes brightness
# Note: this can be run without using superuser terminal
```

`xdg-open` opens file (Linux)

`open` opens file (macOS)

## 1.8 Executable and Unix Shebang

**shebang:** a character sequence involving `#!` at the beginning of a script

- A shebang `#!` indicates that a file is an executable in Unix

```
#!/bin/sh
curl --head --silent https://missing.csail.mit.edu

# First line indicates that program loader should run the
# program /bin/sh, passing path/to/script (name of this file)
# as the first argument.
```