Missing Semester

Lecture 1 (The Shell)

- How do shell knows where programs are stored.
  - Environment Variable
    - Things that are set whenever you start your shell.
  - PATH Environment variable contain a colon(:) Separated list of paths the shell will search for executables.

- `ls [OPTION]... [FILE]...`
  - Square bracket indicates that it is optional.
  - Triple dot means 0 or more

- Permissions: Owner Permission: R W X
  Group Permission: R W X
  Permissions for Every one else: R W X

- Meaning of Permissions for Directory

  Read: Are you allowed to see which files are in that directory.

  Write: Are you allowed to rename/create/delete files in that directory

  Execute: Are you allowed enter in that directory

- Every program by default has two primary streams:

  1. Input Stream
  2. Output Stream

- Shell gives ways to re-wire these streams

  prog > file — Writes the output of the program into the file

  prog < file — prog will receive the content of the file as its input

  prog >> file — Same as prog > file but instead of overwrite, it will append

prog1 | prog2 Out of prog1 is given to prog2 as input

- grep It is a program that let you search in an input stream for given keywords
- tee Takes input from stdin and write it to a file and stdout

Lecture-2 (Shell tools & Scripting)

- Defining variable in bash

  foo=boo

  → Here variable foo is created and value boo is assigned to it.

  → Note: No space, giving space means different thing.

- Defining function in bash

```
mcd () {
    mkdir -p "$1"
    cd "$1"
}
```

{Makes directory & cd into it}

- The source command reads and executes commands from the file specified as its argument in the current shell environment.

  → It is useful to load functions, variables and configuration files into shell scripts.

- `$0` - Name of the script
- `$1` to `$9` - Arguments to the script. `$1` is the first argument and so on.
- `$@` - All the arguments
- `$#` - Number of arguments
- `$?` - Return code of the previous command
- `$$` - Process Identification number for the current script
- `!!` - Entire last command, including arguments. A common pattern is to execute a command only for it to fail due to missing permissions, then you can quickly execute it with sudo by doing `sudo !!`
- `$_` - Last argument from the last command. If you are in an interactive shell, you can also quickly get this value by typing `Esc` followed by `.`

- Exit codes can be used to conditionally execute commands using && (and operator) and || (or operator).

- Commands can also be separated within the same line using a semicolon ;

  false => Returns error code 1
  true => Returns error code 0

- Another common pattern is wanting to get the output of a command as a variable.
  - This can be done with command substitution.
  - Whenever you place $(CMD) it will execute CMD, get the output of the command and substitute it in place.

- A lesser known similar feature is process substitution, <(CMD)
  - will execute CMD and place the output in a temporary file
  - and substitute the <() with that file's name.
  - Can be useful as some commands needs file as input

- 2> is used to write stderr to the file
  - prog > file1 2> file2
    - file1 will contain stdout from prog
    - file2 will contain stderr from prog

- For Loop

  for loops iterate through a set of values until the list is exhausted.

```
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

```
Looping .... number 1
Looping .... number 2
Looping .... number 3
Looping .... number 4
Looping .... number 5
```

- If Statement

```
if [ <some test> ]
then
    <commands>
fi
```

When performing comparisons in bash try to use double brackets [[ ]] in favor of simple brackets [ ].

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

- When launching scripts, you will often want to provide arguments that are similar.
- Bash has ways of making this easier, expanding expressions by carrying out filename expansion.
  - These techniques are often referred to as shell globbing.

Wildcards

- Whenever you want to perform some sort of wildcard matching you can use ? and * to match one or any amount of characters respectively.

Curly braces {}

- Whenever you have a common substring in a series of commands you can use curly braces for bash to expand this automatically.

★ Shell tools

① Finding how to use a command

- The first approach is to call command with the -h or --help flags.
- A more detailed approach is to use the man command.
- Sometimes manpages can be overly detailed descriptions of the commands and it can become hard to decipher what flags/syntax to use for common use cases.
  - use tldr command

② Finding files

- All UNIX-like systems come packaged with find, a great shell tool to find files.

```
# Find all directories named src
find . -name src -type d
# Find all python files that have a folder named test
find . -path '**/test/**/*.py' -type f
# Find all files modified in the last day
find . -mtime -1
# Find all zip files with size in range 500k to 10M
find . -size +500k -size -10M -name '*.tar.gz'
```

- Beyond listing files, find can also perform actions over files that match your query.

```
# Delete all files with .tmp extension
find . -name '*.tmp' -exec rm {} \;
```

or -iname if you want the pattern matching to be case insensitive.

3 Finding Code

grep

- generic tool for matching patterns from the input text
- -C for getting Context around the matching line
- -v for inverting the match, i.e. print all lines that do not match the pattern.
- When it comes to quickly parsing through many files, you want to use -R since it will Recursively go into directories and look for text files for the matching string.

4 Finding Shell commands

- The history command will let you access your shell history programmatically.
  - It will print your shell history to the standard output.
- In most shells you can make use of Ctrl+R to perform backwards search through your history.
- This can also be enabled with the UP/DOWN arrows.

5 Directory Navigation

ls

tree

- Limit the level or depth of recursion ($ tree -L 2)
- Show all files including hidden dot files: ($ tree -a)
- Show only the directories: ($ tree -d)

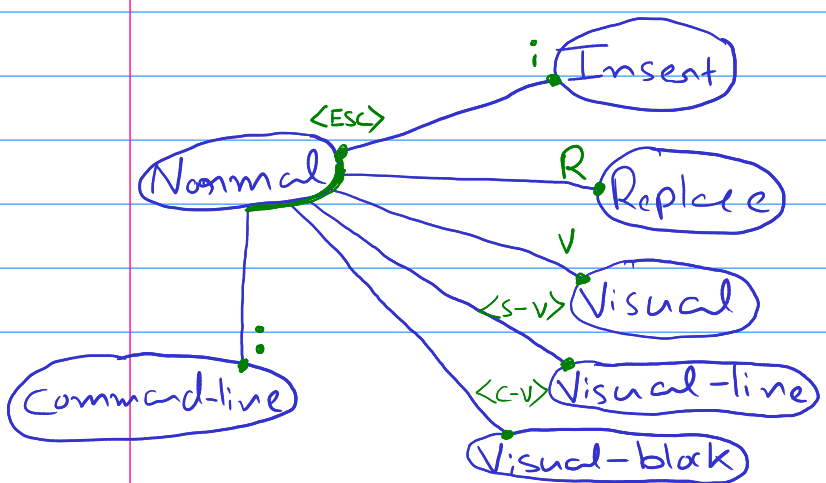
From Exercise

- A file descriptor is nothing more that a positive integer that represents an open file.
- In Unix systems, everything is a file.
- There are file descriptors for the Standard Output (stdout) and Standard Error (stderr).
  - It will always be 1 for stdout and 2 for stderr.
- 1> re-directs output of stdout
  - If FD is not given it defoults to 1. (i.e. 1> is same as >)
- 2> re-directs output of stderr
- 2>&1 redirect fd 2 (stderr) to simply do what fd 1 is doing

Lecture 3 {Editor Vim}

- Vim is a Model Editor.
  - It has multiple operating mode.

Mode
- Noormal mode {Vim starts in this mode}
  - Used for reading & navagating.
- Insert mode
  - For adding text.



Note
^V or Ctal-V or <C-V> means same things

- Vim Comman Commands
  - :q to quit the current window
  - :w to save
  - :wq save and quit
  - :help <command> gives help about the command
  - :qa quit all the open windows

- Normal mode (Navigation)
  - j -> down
  - k -> Up
  - l -> right
  - h -> left
  - w -> right by one word
  - b -> left by one word
  - e -> right keeping the cursor at end of a word
  - 0 -> start of the line
  - $ -> end of the line
  - ^ -> first non empty character of the line
  - ^U -> Scroll up
  - ^D -> Scroll down
  - G -> End of page
  - gg -> Start of page
  - f<char> -> jumps to first <char> in the line
    - example: jf -> jump to first in the line
  - F<char> -> same thing in backward

- Normal mode (Editing commands)
  - i -> Changes mode to insert mode
  - o -> Adds a new line just below, move cursor there and change mode to insert
  - O -> Same as o but above
  - :undo or u
  - :redo or ^r

- d<movement_command> for deleating
  - de -> till end of this word
  - dw -> till start of next word

  {Examples}
- c<movement_command>
  - delete + it puts you in insert mode
- dd -> deleates the line
- cc -> deletes line + put in insert mode
- x -> deleates the particular character
- r<char> -> replaces that charactor with <char>

- Copy and Paste (y(i.e. yank) & p)
  - y takes movement commands y<movement_command>
- Visual mode
  - can be used to select text by using movement commands
  - Visual-line mode and visual-block mode is used to select lines of code and blocks of code respectively
- Doing same thing number of times
  - <number_of_times><command>

    {Example: 4j ⇒ Move down 4 times}
- Modifiers
  - % -> to jump back and fro between prenthesies
  - di(-> delete the content between (), similarly for {} and []
  - da(-> delete the content between () incliding (), similarly for {} and []
- Search
  - /<word> -> moves the cursor to the first place where the <word> is found
  - n -> moves to the next <word>

. -> executes the previous command

4

Data Wrangling

Basic idea is that you have data in one formate and you want it in different formate.

less

→ Opens a pager of the input stream data, and lets you navigate through it easily.

sed — Editor for filtering and transforming text

→ There are tons of commands, but one of the most common ones is s: substitution.

```
sed 's/.*Disconnected from //'
```

→ Replace this patern with nothing.

Regular expressions

→ A powerful construct that lets you match text against patterns.

→ Regular expressions are usually (though not always) surrounded by /.

→ Most ASCII characters just carry their normal meaning, but some characters have "special" matching behavior.

Very common patterns are:

- `.` means "any single character" except newline
- `*` zero or more of the preceding match
- `+` one or more of the preceding match
- `[abc]` any one character of `a`, `b`, and `c`
- `(RX1|RX2)` either something that matches `RX1` or `RX2`
- `^` the start of the line
- `$` the end of the line

⇒ sed's regular expressions are somewhat weird, and will require you to put a \ before most of these to give them their special meaning.
Or you can pass -E.

⇒ sed 's/am/is/g'

→ 'g' modifire to substitite as many times the matern matches.

→ other wise it will only do once per line.

⇒ Well, * and + are, by default, “greedy”.
They will match as much text as they can.

⇒ capture groups

- Any text matched by a regex surrounded by parentheses is stored in a numbered capture group.
- These are available in the substitution as \1, \2, \3, etc.

⇒ By default Regular Expressions match per line.

wc -l

- Counts number of line in input stream.

sort

- Sorts the input stream.

  sort -nk1,1 -> numeric sort on the first column of the input

uniq -c

- Only print lines those are unique
- -c will count the number of duplicates

tail -n5

- prints last 5 lines of the input stream

head -n5

- prints first 5 line of the input stream

awk

- It is a column based stream processor
  (white spaced separated coumn)
- awk '{print $2}' -> this will print the second column from the input stream

paste

- paste -sd, Takes a bunch of line and pasts them together into a single line with delimiter ,

bc -l

↳ Takes a mathematical expression and evaluates it.

5 Command line Environment

★ Job Control

⇒ sleep 20 -> The process will sleep for 20 seconds

⇒ In some cases you will need to interrupt a job while it is executing, for instance if a command is taking too long to complete

⇒ Most of the time, you can do Ctrl-C and the command will stop.

⇒ Your shell is using a UNIX communication mechanism called a signal to communicate information to the process.

↳ For this reason, signals are software interrupts.

SIGINT -> Ctrl-C

SIGQUIT -> Ctrl - \

SIGSTOP -> Ctrl-Z

↳ pauses a process.

⇒ The jobs command lists the unfinished jobs associated with the current terminal session.

⇒ We can then continue the paused job in the foreground or in the background using fg or bg, respectively.

↳ bg %1 -> will resume 1st process in background.

↳ fg %1 -> will take the running process 1 from background to foreground

kill

↳ kill %1 will kill the first process

↳ kill -STOP %1 will stop the first process

Terminal Multiplexers

tmux

tmux expects you to know its keybindings, and they all have the form <C-b> x where that means:
(1) press Ctrl+b,
(2) release Ctrl+b
(3) press x.

Sessions

- A session is an independent workspace with one or more windows
- tmux starts a new session.
- tmux new -s NAME starts it with that name.
- tmux ls lists the current sessions
- Within tmux typing <C-b> d detaches the current session
- tmux a attaches the last session.
  - You can use -t flag to specify sesson.
- Press your prefix <C-b> : and type kill-session, then hit Enter.
- Press <C-d> to close tmux

Window

- Equivalent to tabs in editors or browsers, they are visually separate parts of the same session

`<C-b> c` Creates a new window. To close it you can just terminate the shells doing `<C-d>`
`<C-b> N` Go to the *N* th window. Note they are numbered
`<C-b> p` Goes to the previous window
`<C-b> n` Goes to the next window
`<C-b> ,` Rename the current window
`<C-b> w` List current windows

Panes

- panes let you have multiple shells in the same visual display.

`<C-b> "` Split the current pane horizontally
`<C-b> %` Split the current pane vertically
`<C-b> <direction>` Move to the pane in the specified *direction*. Direction here means arrow keys.
`<C-b> z` Toggle zoom for the current pane
`<C-b> [` Start scrollback. You can then press `<space>` to start a selection and `<enter>` to copy that selection.
`<C-b> <space>` Cycle through pane arrangements.

★ Aliases

⇒ It can become tiresome typing long commands that involve many flags or verbose options.

⇒ For this reason, most shells support aliasing.

⇒ A shell alias is a short form for another command that your shell will replace automatically for you.

↳
```
alias alias_name="command_to_alias arg1 arg2"
```

★ Remote Machines

⇒ It has become more and more common for programmers to use remote servers in their everyday work.

Secure Shell (SSH)

⇒ To ssh into a server you execute a command as follows:

↳ ssh username@machine-name

↳ machine-name will be converted to IP address by DNS.

⇒ An often overlooked feature of ssh is the ability to run commands directly.

↳ ssh foobar@server ls will execute ls in the home folder of foobar.

SSH Keys

↳ Key-based authentication exploits public-key cryptography to prove to the server that the client owns the secret private key without revealing the key.

⇒ To generate a pair you can run ssh-keygen

```
ssh-keygen -o -a 100 -t ed25519 -f ~/.ssh/id_ed25519
```

Key based authentication

↳ ssh will look into .ssh/authorized_keys to determine which clients it should let in.

⇒ To copy a public key over you can use:

cat .ssh/id_ed25519.pub | ssh foobar@remote 'cat >> ~/.ssh/authorized_keys'

or

ssh-copy-id -i .ssh/id_ed25519.pub foobar@remote

SSH Configuration

→ ~/.ssh/config.

```
Host vm
    User foobar
    HostName 172.16.174.141
    Port 2222
    IdentityFile ~/.ssh/id_ed25519
    LocalForward 9999 localhost:8888
```

Port Forwarding

→ Local Port Forwarding

→ Remote Port Forwarding

8

Meta programming

⇒ Semantic Versioning

8 · 1 · 7

major change number is incrimented when we make a backward incompatable change

Majoon

Minoon

Patch

If change we made is entirly backward compatable we in patch change number

I we add new feature we increment minor change number and set the patch to zero.

Deamons

- These are processes that are always running in the background rather than waiting for a user to launch them and interact with them.
- Daemons often end with a d to indicate so
  - For example sshd, the SSH daemon, is the program responsible for listening to incoming SSH requests and checking that the remote user has the necessary credentials to log in.
- In Linux, systemd (the system daemon) is the most common solution for running and setting up daemon processes.
- You can run systemctl status to list the current running daemons.
- Systemd can be interacted with the systemctl command in order to enable, disable, start, stop, restart or check the status of services.