Sysadmin DeCal

Toggle Dark Theme

Lab 2 - Core Shell & Shell Scripting

Facilitator: Joey Li, Aaron Zheng

11 min read

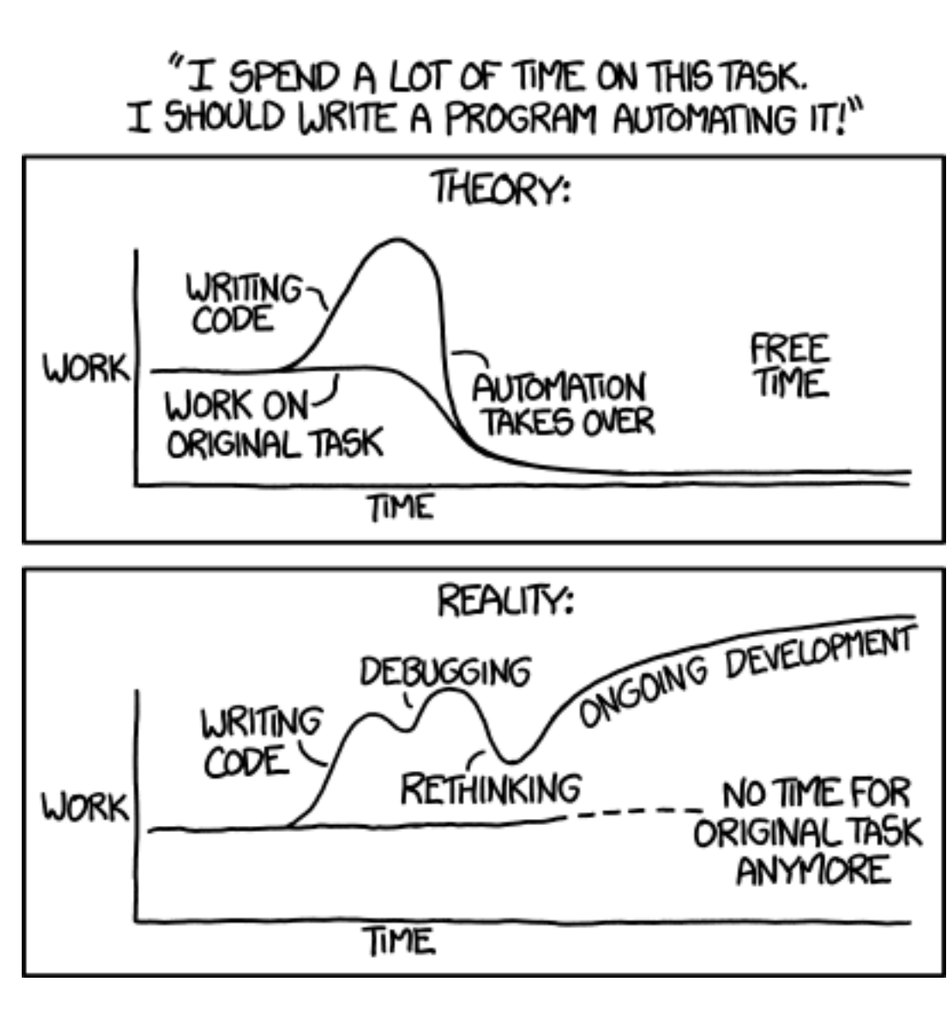
TABLE OF CONTENTS

1 Setting up
2 A quick intro to `vim`
  a Why `vim`?
  b Hello World
  c The `vim` Modes
    a Normal mode:
    b Insert mode:
    c Visual mode:
  d Questions
3 A quick intro to tmux(Optional)
  a Why tmux?
  b Getting Started:
  c Questions (Optional)
4 Scripting
5 Why Scripting?
6 Scripting Lab Assignment
7 Skeleton Code
8 Some tips to make things easier
9 Submitting the lab
10 Additional Resources

Welcome to Lab 2! In this lab you will be learning how to work productively in a shell and use that to write your first shell script.

Remember to submit your answers in the Gradescope assignment!

Don't forget to use Google and `man` when stuck. The resources linked at the bottom may be helpful as well.



Setting up

This lab can be completed either on tsunami by logging in using ssh or on the VM you set up in the previous lab. This lab requires a bash shell, `vim`, and `tmux`. If you do not have tmux or vim installed: `sudo apt install vim tmux`

A quick intro to `vim`

`vim` is a very widely used text editor. It's well known for its customizability and plethora of keybinds. While it may be somewhat unintuitive to use at first (since a lot of common keybinds for things like copy-paste, saving, or exiting don't do what you think they will), it's well worth learning about, and you'll certainly come across it all the time when working in the shell!

Why `vim`?

- It's a descendant of vi, which was written in Berkeley by Bill Joy, who went on to found Sun Microsystems.
- Sometimes you will be suddenly thrown into `vim` via merging git conflicts or other programs.
- It's included in practically every UNIX environment.
- You can be very productive when familiar with it.

Hello World

To get started with learning `vim`, run the command `vimtutor`. This will walk you through the below material in an interactive manner! You aren't required to finish the entire tutorial, but we encourage you to at least complete lesson 1. You can then use this section as reference if you forget anything.

The `vim` Modes

Vim is a modal text editor, meaning that you can change editing modes in order to do different things. There are 3 primarily used modes: **Normal**, **Insert**, and **Visual** mode.

NORMAL MODE:

- Used for moving around and performing actions
  - `hjkl` to move left, up, down, and right (arrow keys work too but hjkl is usually faster to use!)
  - `G` to move to end of file, `gg` to move to beginning
  - `i` to enter **i**nsert mode (`a` , `o` also change mode in different ways)
  - `dd` to cut a line
  - `yy` to copy a line
  - `p` to paste
  - `/` to search
  - `u` to undo
- Type in commands with `:`
  - Save with `:w`
  - Exit with `:q`
- Explore more commands online! Here's a cool cheat sheet to get you started.

INSERT MODE:

- Used for editing text like a usual editor
- Arrow keys to move
- Esc to exit to normal mode (lots of people bind it to Caps Lock)

VISUAL MODE:

- Enter with `v` from normal mode
- Used to select text visually
- Modify selection with normal mode movement commands
- Use `o` to move the cursor to the other side of the selection
- Yanking, deleting, and pasting use `y` , `d` , `p` (sound familiar?)

A key feature of `vim` is **chaining together commands**. Normal mode is essentially a massive amount of shortcuts that you can combine to quickly navigate and edit a file. Want to move down 3 lines? You know that `j` means move down 1 line, so you can use `3j` to move down 3. `d` is for deletion and `w` is to jump to the next word, so what does `dw` do?

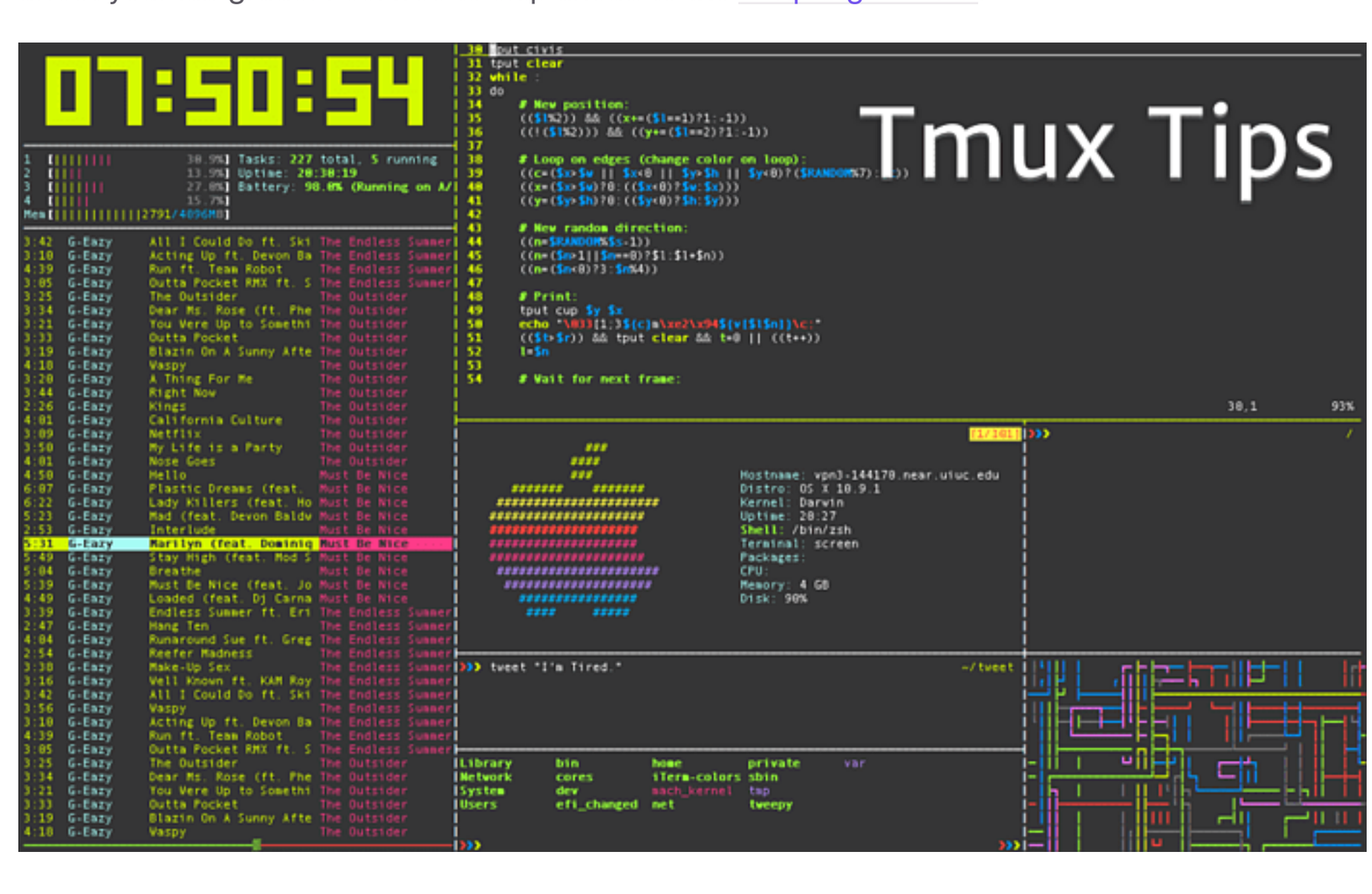
Questions

Try playing around with lab2.md while looking up some new commands. Use `wget` to download it!

1. How would you delete the previous 10 lines?
2. How would you jump back to the shell without exiting `vim`?
3. How would you edit a new file alongside another file?
4. How would you indent a block of text?
5. Tell us about one other cool vim feature you found out about that isn't mentioned in this lab!

A quick intro to tmux(Optional)

While we reccomend that you complete this section of the lab, it's completely optional and will not affect your lab grade. Feel free to skip ahead to the Scripting Section.



Why tmux?

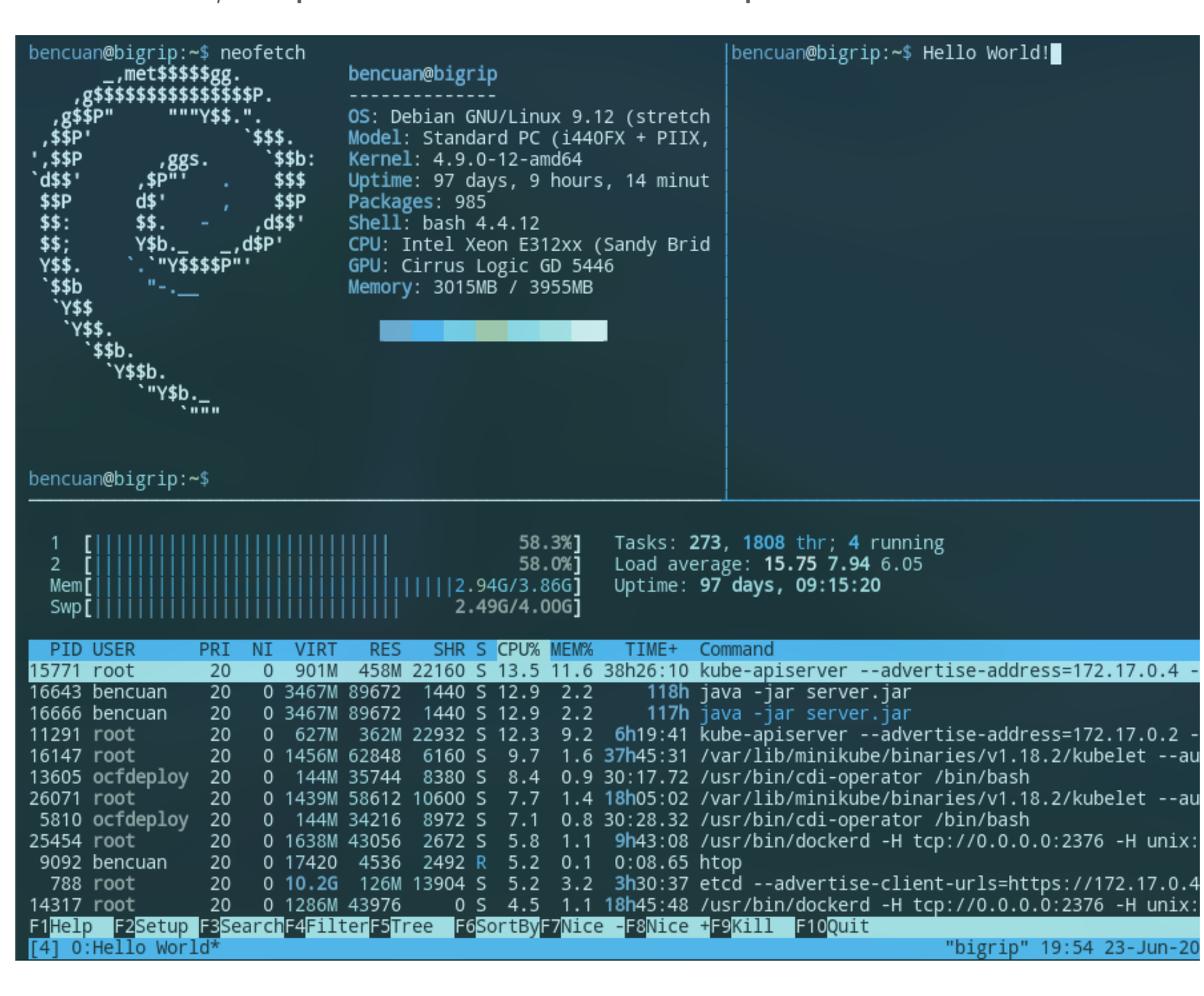
- You can open multiple windows when sshed into a machine.
- You can go compile and run programs while editing them.
- You can logout and ssh back in without having to reopen all your files.

Getting Started:

- Start a session with `tmux`.
- Detach from a session with `Ctrl-b d` (press d after releasing `Ctrl-b`)
- Split into 2 panes with `Ctrl-b %` (vertical) or `Ctrl-b "` (horizontal)
- Swap current pane with `Ctrl-b o`
- Find more information about tmux online. You might find [this cheat sheet][Tmux cheat sheet] helpful!

Questions (Optional)

1. Make a new tmux session. Using tmux shortcuts, try to make your session have a similar layout to the one below, and **upload a screenshot of it to Gradescope!**



Some things to note:

- The top left panel is resized. By how much, it doesn't matter.
- The top right panel is named "Hello World". (You can see this name displayed on the bottom left.)
- You don't need to run any of the commands I did, but they do look pretty cool :) Try to figure out what command the bottom panel is running, and what it does!
- Don't worry about copying the layout exactly. The purpose of this exercise is simply to help you get comfortable making custom layouts in tmux.

1. If you haven't already, detach from your current tmux session using `Ctrl+b d`. Now, what command would you type to attach back to it?
2. What command will delete your session?
3. What command will create a new session?

Scripting

Why Scripting?

Many of the tasks that someone would like to perform on a computer are regular, require repetition, or are menial or tedious to do by hand. Shell scripting allows one to interact programmatically with a shell to do certain tasks. For example, the command for scanning log files in the previous topic guide could be automated to be performed on a schedule by means of a shell script. `bash` scripts are an incredibly powerful tool for sysadmins to automate tasks that are otherwise difficult to remember or long-running.

In cases where shell syntax is inappropriate for the task at hand, one can instead call into programs written in other languages, such as Python, which can read from stdin, process data, and write to stdout.

Before you jump into the assignment we reccomend reading through our Scripting Reference which provides a introduction to Bash and Python(optional part in this lab) scripting in Linux. Feel free to come back to this reference as you work through the assignment.

Scripting Lab Assignment

You'll be completing a classic first shell scripting assignment: make a phonebook.

Write a shell script `phonebook` which has the following behavior:

- `./phonebook new <name> <number>` adds an entry to the phonebook. Don't worry about duplicates (always add a new entry, even if the name is the same).
- `./phonebook list` displays every entry in the phonebook (in no particular order). If the phonebook has no entries, display `phonebook is empty`
- `./phonebook remove <name>` deletes all entries associated with that name. Do nothing if that name is not in the phonebook.
- `./phonebook clear` deletes the entire phonebook.
- `./phonebook lookup <name>` displays all phone number(s) associated with that name. You can assume all phone numbers are in the form `ddd-ddd-dddd` where `d` is a digit from 0-9.
  - **NOTE:** You can print the name as well as the number for each line. For an additional challenge, try printing all phone numbers *without* their names. (See the example below for more details)

For example,

```
$ ./phonebook new "Linus Torvalds" 101-110-0111
$ ./phonebook list
Linus Torvalds 101-110-1010
$ ./phonebook new "Tux Penguin" 555-666-7777
$ ./phonebook new "Linus Torvalds" 222-222-2222
$ ./phonebook list
Linus Torvalds 101-110-1010
Tux Penguin 555-666-7777
Linus Torvalds 222-222-2222
# OPTIONAL BEHAVIOR
$ ./phonebook lookup "Linus Torvalds"
101-110-1010
222-222-2222
# ALTERNATIVE BEHAVIOR
$ ./phonebook lookup "Linus Torvalds"
Linus Torvalds 101-110-1010
Linus Torvalds 222-222-2222
$ ./phonebook remove "Linus Torvalds"
$ ./phonebook list
Tux Penguin 555-666-7777
$ ./phonebook clear
$ ./phonebook list
phonebook is empty
```

If you run into an edge case that isn't described here, you can handle it however you wish (or don't handle it at all). You can assume all inputs are in the correct format.

Skeleton Code

To help you in this task, skeleton code for this lab can be found here. Once you are done with this task, you can submit your work on Gradescope.

As an optional (but recommended) assignment: Try implementing the same Phonebook behavior, but in `python`! This will highlight some of the strengths and weaknesses between the two languages. If you're already familiar with `python`, you may find it helpful to do this before implementing it in `bash`.

Some tips to make things easier

- `bash` has an append operator `>>` which, as you might guess, appends the data from its first argument to the end of the second argument.

```
$ cat foobar.txt
foobar
$ echo "hello, reader" >> foobar.txt
$ cat foobar.txt
foobar
hello, reader
```

- `bash` also has a redirect operator `>`, which takes the output of one command and outputs it to a file.

```
$ cat foobar.txt
foobar
$ echo "hello" > foobar.txt
$ cat foobar.txt
hello
$ > foobar.txt
$ cat foobar.txt
$
```

- Remember that you can simply write to and read from a file to persist data.
- In bash, changing lines can be done through the `sed` command. If you wish to do so, the format `sed -i "s/<old>/<new>/g" ./filename` may be helpful in this lab (e.g. for deleting a line or part of a line). For example:

```
$ echo "hello 123" > foobar.txt # writes hello to foobar.txt
$ cat foobar.txt
hello 123
$ sed -i "s/h/j/g" foobar.txt
$ cat foobar.txt
jello 123
# You can also use regex: learn more at regex101.com
$ sed -i "s/[0-9]\{3\}/world/g" foobar.txt
$ cat foobar.txt
jello world
```

- Recall that `bash` exposes its command line arguments through the `$<integer>` positional parameters:

```
#!/bin/bash
# contents of argscript.sh

echo "$1"
echo "$2"
```

```
$ ./argscript.sh foo bar
foo
bar
```

- In `bash`, single quotes `''` preserve the literal value of the characters they enclose. Double quotes `""` preserve the literal value of all characters except for `$`, backticks `` ` ``, and the backslash `\\`. The most important implication of this is that double quotes allow for variable interpolation, while single quotes do not. You can think of single quotes and the stronger "escape everything" syntax while double quotes are the more lax "escape most things" syntax.

```
$ echo '$LANG'
$LANG
$ echo "$LANG"
en_US.UTF-8
```

- In `python`, you can interact with command-line arguments through the `sys.argv` list

```
# !/usr/bin/python
# contents of argscript.py

import sys
print(sys.argv[1])
print(sys.argv[2])
# end of file
```

```
$ ./argscript.py foo bar
foo
bar
```

- `python` lets you manipulate files with the `open` function, commonly used with the `with` control structure

```
# !/usr/bin/python
# contents of fileman.py

with open('./newfile.txt', 'w') as f: f.write("hello from python\n")
# end of file
```

```
$ python fileman.py
$ cat newfile.txt
hello from python
```

- If you are getting permission denied issues, you will probably need to make your `phonebook.sh` executable: `chmod +x phonebook.sh`

Submitting the lab

Once you're done remember to submit your answers to Gradescope. There are multiple valid answers for some of the questions.

For the scripting assignment, you will need to upload a file containing your script. If you edit and test your script in your VM using `vim` or another shell editor, you can copy the file to your local machine using `scp` or simply copy-paste the text into a local file.

Don't be stressed about getting something correct; just have fun exploring.We'll release the answers after the lab is due!

Additional Resources

Keybindings

Learning vim progressively

Tmux cheat sheet

Thanks to Nvidia for sponsoring GPUs to the OCF.

Huge thanks to Linode for sponsoring the equipment used to record digital lectures for the Decal

Hosted by the OCF