Week 08 Homework 💻 🦄

New stuff learned this week:

- the `~/.bashrc` file is like *preferences* for your bash shell — all the code in it gets run every time you start up bash
- I snuck a line into your `.bashrc` that puts bash into *vi mode,* which makes it work mostly like our beloved `vim` — you can hit `<ESC>` to get into *normal mode* and use most `vim` features to edit your shell commands
- *variables* in bash are created by typing `FOO=bar` where `FOO` is the variable *name* and `bar` is the *value.* You can also do `FOO="Jared is cool"` — using double-quotes to surround a string that has *spaces* in it.
- *variables* in bash are ***accessed*** by using `$<variable-name>` like `echo $FOO` , and variables can be used alongside other text inside a string, like `echo "I like $FOO"`
- there are some *built-in shell variables* you can access, like `$HOME` `$SHELL` `$USER` `$OSTYPE`
- every bash command ***exits*** with a *code* of either `0` for success 🍾 , or `1` (or another non-zero number) for failure 💔 .
- you can access the exit code of the last command being run with a special variable called `$?` — like `echo $?`
- you can make a simple ***bash script*** by creating a file with lines of bash commands in it, then you can *execute* (meaning "run") that script by typing `bash <path-to-script-file>` — by convention, shell scripts are given the extension `.sh`
- the `sed` command is a ***stream editor*** — it allows you to make changes line by line to text files as you're piping them to and from things. The basic syntax is `sed -E 's/<regular-expression>/<replacement>/<optional-flags>'` — so for instance `sed -E 's/goat/chicken/'` would replace the word "goat" with "chicken" 🐐 > 🐔
- Regular Expressions are like a "mini-language" that is used by almost all computer languages and many shell commands, they are used to find and sometimes replace patterns in text files.
- Regular Expressions make use of ***meta-characters*** to represent concepts, giving a lot of power and flexibility
  - the `.` metacharacter means "match any character"
  - the `^` metacharacter means "match the beginning of the line"
  - the `$` metacharacter means "match the end of the line"
  - the `*` metacharacter means "match ***zero*** or more of the character before it"
  - the `+` metacharacter means "match ***one*** or more of the character before it"
  - the `?` metacharacter means "the character before it is ***optional***"

Touch Typing Links:

- http://touchtype.co
- https://www.how-to-type.com

Homework plan:

- 1 day creating more flash cards
- 1 day reviewing all flash cards
- 2 days CLI practice
- 1 day `vim` practice
- 2 days touch-typing practice
- watch CCCS#7 — definitely twice

Homework day 1:

- do flashcard assignment (see below)
- touch typing practice

Homework day 2:

- CLI practice #1
- watch CCCS#7

Homework day 3:

- touch typing practice
- `vimtutor` — Everything *except* Lesson 7

Homework day 4:

- CLI practice #2
- watch CCCS#7
- review all your flash cards

Flashcard assignment

First, if you didn't make your flashcards from last week, ***go back and do them***. Then:

1. spend a minute or two reviewing your flash cards
2. on the cards you've already made, add the word `SHELL` in the upper left corner — this is to add a *context*, so that you don't get confused later when there are things like how `.` means one thing

in the shell (my current directory) and another thing in regular expressions (match any single character).

3. Make a small pile of new flash-cards, with the word `REGEXP` in the upper left corner (short for Regular Expressions), make one card for each of the following meta-character symbols we went over. Put the definition on the back of the card:
   a. `^`
   b. `$`
   c. `.`
   d. `*`
   e. `+`
   f. `?`

CLI Practice #1

1. carefully review the "new stuff we learned this week" section above
2. `ssh` into your home dir
3. make a new directory called `week8` and `cd` down into it
4. Make a bash variable called `NAME` that contains your first name
5. use the `echo` command to print your variable to standard out
6. use the `echo` command and your variable to print `My name is <yourname>` to standard out.
7. Make another variable called `FULLNAME` that contains your first, middle, and last name, separated by spaces. (hint, you'll need to use double-quotes for this one)
8. use both of your variables to print to standard out the phrase: `My name is <your-full-name>, but you can call me <first-name>!`
9. Redo the last command, but this time, send the sentence into a file called `name.txt`
10. Use one of the special, built-in variables bash gives you to `echo` the sentence `The path to my home directory is <home-dir-path>.`
11. Redo the above command, but this time ***append*** the text to the `name.txt` file.
12. Type another command, using another built-in variable, that will *append* another sentence to the same file that reads: `My username is <your-user-name>.`
13. Rename the `name.txt` file to `about-me.txt`
14. Type a command to print the contents of `about-me.txt` to standard out, it should have 3 lines of text on it.
15. Open `about-me.txt` in `vim` and add *two new lines* to the end: one saying `My favorite color is <your-fav-color>.` and another saying `My birthday is <your-birthday>.` Then, save and close the file.
16. `cat` out the contents of `about-me.txt` and pipe the result into `sed` , using `sed` to replace your first name with the name "Bob". You should see your file's contents sent to standard out, with the name "Bob" swapped in for some places.
17. Redo the above command, but this time pipe it *again* to `sed` and use `sed` to replace the word `is` with `is not` on each line. It should now say strange things like "My name is not Bob" and "My favorite color is not blue", etc.
18. Redo the above command, but this time redirect the ouput into a file called `false.txt`

19. Type a command that will print the *exit code* of the last command to standard out.
20. Type a command that will produce an error, and then print out the exit code — it should be `1`
21. Use `echo` and your knowledge of variables to print the following message to standard out: `The last command exited with the code <exit-code>.`

CLI Practice #2

1. `ssh` into your home dir
2. mentally review how we talked about using `<ESC>` and ***vi mode*** in the shell, to edit shell commands in a vim-like manner. As you do the exercises below, try to start practicing using vim commands to edit and move around within your commands.
3. use `ls` and an **absolute path**, list out the contents of the directory one above you (you can't use `..` — use an absolute path)
4. now use `ls` again, with an absolute path, to see what's inside the `regex` folder, which you should have seen in step 2
5. now, using the `cp` command, copy both the `goats.txt` and `chickens.txt` files from where you found them, *down into the* `week8` folder you made during practice #1. You should still be in your home dir, and you can use *relative paths* for this step.
6. now change your directory down into `week8` dir
7. `cat` out the contents of `goats.txt` so you can see whats inside
8. pipe the contents of `goats.txt` into `sed` and use a regular expression to change the word `goats` to `rabbits` - but ***only*** on the second line, the one that says `I like goats`
9. Redo the above command, but change it slightly so that it also changes the third line, so it becomes: `Do you like rabbits?` — if you do it right, only the second and third lines will be changed.
10. Now, use a different regexp to change the word `Goat` (with a capital G) to `Bear` on the first and last lines, but it should not change the word `Goat` on the line that says `…a Goat named Josephus`
11. Now, use `sed` to change every instance of the word `goat` in the whole file to `whale` — Make sure you get the ones with a capital `G` and a lowercase `g` (the replacements can all be lowercase `whale` - you don't need to worry about getting `W`'s)
12. Redo the last command, but change your regular expression so it also fixes where I misspelled `goat` as `groat` on the last line.
13. Make a new regular expression that changes the word `goatpack` AND `llamapack` to `beaverpack` . (hint: you'll probably need to match the word `to ` before the `goatpack` and `beaverpack`)
14. Make up your own regular expression that changes the text in some way, and redirect the output to a file called `silly.txt`
15. copy your `silly.txt` file into my home dir (ubuntu), but change the name while you're copying it to `<yourname>-silly.txt`
16. `cat` out the `chickens.txt` file so you can read what's in it.
17. pipe the contents of `chickens.txt` into `sed` , transforming every instance of `Chicken` , `chickens` or `chicken` or `chickens` into `wombat` (hint, if you add `i` as a *flag* it makes the regex case-insensitive, like this `sed -E 's/foo/bar/i'`) (hint #2 — the pipe character `|` *is itself a*

metacharacter that we haven't learned yet, so if you want to use it in your regexp to stand for the literal character, you need to *escape* it by adding a *backslash* before it, like this `sed -E 's/fo\|o/bar/'`)

18. repeat the last step, but modify your regexp so that it also catches the misspellings on line 2 and 3
19. use `sed` with a new regexp so that the *last line of the file* becomes `Your DOG are DOG full of DOG, DOG them out, OK DOG?` (it's ok if your regexp changes other lines too, just focus on the last line).
20. **Extra Credit** ✨ **—** move into *your* home dir, and then, using the `moby.txt` file in the `/home` dir (one dir above you), use a combination of `head` , `tail` and `sed` (or multiple invocations of `sed`) to isolate a paragraph and transform the text in a funny way. When you're done, paste the command to produce it into Slack so we can try. We should be able to execute the command from our home dir and see the same thing you saw. Don't paste the silly text, paste the command that produces it.
21. **Extra Credit** ✨ — You can assign multiple bash variables in a row, like `NAME=Bob COLOR=blue` and you can then execute a command immediately after, like `NAME=Bob COLOR=blue echo "I am $NAME, I like the color $COLOR"` . With this knowledge, make a shell script called `madlib.sh` that echos out a sentence or two using variables. Write it so that other people can run a command like this, and have a "madlibs"-like experience: `ANIMAL=goat PLURAL_NOUN=fruits PROPER_NAME=Josephus bash madlibs.sh` When you're done, post in slack where to find your file, and what variables we need to fill in to make it work. 👍