Basic 3
Unix and the Shell

EECS 201 Fall 2021

Submission Instructions

Answer the bolded text that begins with a "**Q**". This assignment is an "online assignment" on Gradescope, where you fill out your answers directly.

Preface

This assignment should be fine on Linux and macOS. That being said, if you run into issues on macOS, like a command completely not running as expected, try using the course server or an Ubuntu 20.04 VM.

1 Redirection

Recall from lecture that we can *redirect* the inputs and outputs of a command. `echo` is a command that will print out a string (taking into account any expansions you embed in it). For example, I could use `echo "Welcome $HOME"` to print out `Welcome /home/brandon`. Try modifying this command so that it saves that output to a file.

Suppose I had an application `app` that *read data from standard input*, and produces its *output on standard output* and reports *error messages on standard error*. Let's say we have a file called `input.dat` in the current directory. Assume that `app` is in the `PATH`.

Q: Write a command that feeds `app` data from `input.dat`, saves the output to `output.dat`, and saves the error messages to `log.txt`

2 Pipeline

Recall from lecture that commands can be chained together to form a pipeline, where one process passes its output to the input of the next. For example, I could create this pipeline `cat file1 file2 file3 | rev` to concatenate three files and output the result with `cat` and then reverse the output of `cat` with `rev`.

Q: Using only the commands below, create a pipeline that creates a sorted list of the user's currently running *commands* with no repeats. As you do this question, try testing out your pipeline by running it.

- `awk '{print $11}'`: This will print out the 11th column of each line of input
- `ps ux`: This will list out all of the user's processes. The 11th column is the command column. The first line of output is the header line, providing labels for each column. **You probably wouldn't care about this header line**.
- `sort`: This will sort inputted lines and output them in a sorted order
- `tail -n +2`: This will print the input from the second line onwards (`tail` prints the ends of file; with these arguments we can specify where to start printing from)
- `uniq`: This will take inputted lines and omit repeated lines: for example if "hello world" occurs on three **adjacent** lines, only one "hello world" is outputted.

3 Variables and expansion

Recall that there's two kinds of variables: environment variables, part of the "metadata" for each process, and shell variables, variables that live inside the data for the shell program itself.
To use them, we use "expansion" to replace instances of `$VARNAME` with the value of the variable (environment or shell) in a command before it gets executed. There's also "command substitution" which allows us to do a similar thing, except with commands instead of variables. For example, `echo $(ls)` could result in the following expansion `echo file1 file2 file3`.

3.1

Q: Suppose you wanted to print out the value of the variable `PATH`. What command would you type out and run?

3.2

Q: Write a command that saves the output of `git status` to *shell* variable `gs`

4 Control flow

I want to re-emphasize that the conditionals for the if-elif-else blocks, while loops, and until loops are *commands*: they aren't any special kind of syntax. Don't believe me? Try running this Bash conditional command: `$ [[ 100 -gt 200 ]]`. This should have a exit status of 1, as 100 is actually less than 200; try checking this exit status.
As you do this part, remember that there's no difference between writing these blocks in a script or entering them in line-by-line: you can test things out at the command-line.

4.1

Suppose that a command `cmd` has just run, thus setting the exit status `?` variable. We will be writing an if-elif-else block to check what `cmd` exited with. Note that if-elif conditions are based on the exit status `?`: special conditional commands like `[ ]` and `[[ ]]` are technically commands that set `?` as conditions are checked, so we need to save the exit status of our target command `cmd` to check against.

Q: Set a shell variable `status` to save the exit status `?`. Next write an if-elif-else block `echo`es `SUCCESS` if the exit status of `cmd` was 0, `FAILURE` if the exit status was 1, and `ERROR` otherwise.

4.2

The `seq` utility prints out a list of numbers.

Q: Write a for-loop using a *command substitution* with `seq` that uses `echo` and an output redirection `>` to create and write `hello` to 25 files, starting with `file1.txt` and ending with `file25.txt`. As a result, each of these files should contain the string `hello` inside of it.

Controlling your environment

This section is ungraded and is more for your personal edification. Many shells have a configuration file that gets sourced when an instance of the shell starts (i.e. when the instance starts up, it executes all commands in that file). For example Bash has its `.bashrc` and Zsh has its `.zshrc`.

Recall that the `PATH` variable is a colon-delimited list of directories to look for executables to run.

Q: How would you add a directory `$HOME/scripts` to your `PATH` so that each new instance of your shell has this updated `PATH`?

Another utility of shells is `alias`, which allows you to create aliases for commands. Such aliases could serve as shorthand for some commonly used commands and their arguments. For example, I have an alias on my WSL that `cd`s to my Windows user directory. Try playing around with it!