Advanced Exercise – Week 3

Due: Before October 8, 10:00PM

Submission Instructions

To receive credit for this assignment you will need to stop by someone's office hours, demo your running code, and answer some questions.

1 Pretty `PS1`

Open a new terminal and try the following commands in order:

```
1  echo -e "\\033[44;3;70;38;5;214m"
2  <hit enter again>
3  PS1="Hello World -- "
4  echo -e "\\033[44;3;70;38;5;214m"
5  pwd
6  ls
7  pwd
8  echo -e "\\033[44;3;70;38;5;214m"
9  ls --color=none
10 pwd
11 reset
12 pwd
13 bash --norc
14 echo -e "\\033[44;3;70;38;5;214m"
15 ls
16 pwd
17 ls --color=auto
18 pwd
19 exit
20 PS1="\\033[44;3;70;38;5;214mHello Again -- "
21 ls
```

What happened to your terminal as you ran these commands? Play around with some other forms of `PS1` and other colors, see what happens. Why does calling `ls` sometimes reset things?

Create a custom `PS1` for yourself. Look into some of the options for `PS1`, you will need to explain why you added the options you did and decided against options you didn't choose.

Extend your PS1 by writing a bash function the changes your prompt in a way that is not built-in to bash. Some examples: Add an asterisk if your are in a git repository with uncommitted changes. Change the color if you are currently in a shared directory (i.e. in a Dropbox folder). Change the color if the current directory will not be saved across a reboot (i.e. if you are you are somewhere in the `/tmp` directory).

Submission checkoff:

- ☐ Explain what `PS1` does
- ☐ Explain what you did to customize `PS1` and **why** you chose the customizations that you did.
  - ☐ Explain how your custom function works.
- ☐ Explain what the `PS2` variable controls. Change `PS2` from the default and show an example.
- ☐ Type `set -x`. Then type `ls`. Explain all of the output.

2 Understanding tab completions

Open a new terminal and try typing the following (note <tab> means press the tab key):

```
1 p<tab><tab>
2 y
3 <ctrl-c>
4 pi<tab><tab>
5 pin<tab><tab>
6 ping<tab><tab>
7 ping <tab><tab>
8 PATH=<enter>
9 p<tab><tab>
```

In addition to finding programs, tab completions can help you to use a program correctly by hinting at what arguments a program accepts, try this:

```
10 # Open a new terminal (or manually set your PATH correctly again)
11 ping <tab><tab>
12 ping -<tab><tab>
13 ping -I<tab><tab>
14 ping -Q<tab><tab>
15 ping -Q 0 <tab><tab>
```

Today, most programs include tab completion support, but this is a remarkably manual process. Check out the contents of the `/usr/share/bash-completion/completions/` directory.

Now take a look at `/usr/share/bash-completion/completions/ping`. There's a lot going on in this example, but try to see if you can understand some of how the completions are working. What do you think the result of `ping -T <tab><tab>` will be?

(Hint: `||` in bash means "if the previous thing failed, do the next thing". What's the output of `echo $OSTYPE`? Will that equality pass or fail?)

Submission checkoff:

- ☐ Why is the list of tab completions different between lines 1 and 9?
- ☐ What is the default tab completion behavior for a program if no custom completion function has been written?

3 Testing Made Easy

When working on a project in EECS, you should be writing test cases to make sure your program is functioning properly. Checking your test cases can be tedious, but fortunately for us, scripting can help make it really easy to run your test cases and report which ones are passing and which ones are failing. Here, we will do just that.

Go ahead and download some files from this URL, extract them, and take a look. You should see 3 files. `testPass.sh`, `testFail.sh`, and `testTimeout.sh`. You should not need to modify their contents.

```
> wget https://c4cs.github.io/static/w17/advanced/wk3-advanced-p3-files.tar.gz
> # eXtract Ze Files
> tar xzf wk3-advanced-p3-files.tar.gz
```

Write a few more shell scripts (2 or 3) with the string "test" somewhere in the name. These should run a program of your choice and test the output to make sure the program is passing the test case. The program's output can be as simple as "Hello, World!", but the test case must correctly check the output of the target program and signal whether the target program has passed or failed.

Then, write a shell script which takes all files in the current directory with "test" somewhere in the name, runs each of them, and reports whether the program passed or failed the test case by printing "PASSED", "FAILED". Your script should also stop programs from runnning for more than 3 seconds and print "TIMEOUT". Your individual test case scripts should not print anything, but the test runner script should.

Submission checkoff:

- ☐ How does the `exit` command work? (Hint: How is this similar to `return` in C/C++ ?)
- ☐ How can a test case report to our test running script whether the target program passed or failed?
- ☐ How can you tell if a program is running too long?
- ☐ Can you demo your code?