Advanced Homework 2
Shell Vanity or Utility?

Assigned: Friday, January 15, 11:00AM

Due: Before the end of the final office hours this week (Thursday, ~8:15PM)

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:

```
echo -e "\\033[44;3;70;38;5;214m"
<hit enter again>
PS1="Hello World -- "
echo -e "\\033[44;3;70;38;5;214m"
pwd
ls
pwd
echo -e "\\033[44;3;70;38;5;214m"
ls --color=none
pwd
reset
pwd
bash --norc
echo -e "\\033[44;3;70;38;5;214m"
ls
pwd
ls --color=auto
pwd
exit
PS1="\\033[44;3;70;38;5;214mHello Again -- "
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: Include the current branch in the prompt if you are currently in a git repository. 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

This corner of the world is a little rougher and more complex. The goal of this task is to show you how you can use and even do some basic hacking on a tool that you don't completely understand. You ***do not*** *need to develop a deep and complete understanding of bash completions for this task, you simply need to generate something that works.*

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?)

Writing our own completion file

Completions are *really* nice as a user. Find a project you've written for a previous or current EECS class. What happens if you try to tab-complete that project (e.g. `./project1 <tab><tab>`)?

Let's try to make that more useful. Find a current or previous project that takes at least two options (if you don't have one, write a trivial program, such as `mycalc [--add,--subtract] NUM1 NUM2`). Be sure your project executable is named something unique (e.g. there is already a built-in `calc` program).

Write a custom completion file for your program that completes its arguments. Install your completion such that it works whenever you open a new terminal.

For this part, I recommend not starting from scratch. Find a simple completion file[1] *to start from, hack it, mess with it, whatever until it does what you want.*

Submission checkoff:

- ☐ Why is the list of tab completions different between lines 1 and 9?
- ☐ What is the default tab completion behavoir for a program if no custom completion function has been written?
- ☐ Demonstrate your custom completion working.
  - ☐ Explain what you had to do to "install" your completion file
  - ☐ Explain what you had to do to get your completion file working

[1] Preview of coming attractions, try `wc -l /usr/share/bash-completion/completions/* | sort -n | head`