Select the right page in Gradescope or we will not grade the question!

Homework 3
Shells, Environment, and Scripting

Due: Saturday, October 1, 10:00PM (Hard Deadline)

Submission Instructions

Submit this assignment on Gradescope. You may find the free online tool PDFescape helpful to edit and fill out this PDF. You may also print, handwrite, and scan this assignment.

1 Understanding your `PATH`

In a terminal, type `PATH=` (just hit enter after the equal sign, no space characters anywhere). Try to use the terminal like normal (try running `ls`). What happened?

Give an example of a command that used to work but now doesn't:

Trying to use vi as a text editor no longer works. Additionally, I can't use touch to create any new files anymore.

Can you still run this command with an empty `PATH`? How?

One way we could do this is by finding the entire path of the command that we're trying to run. For example, if our PATH Variable was normally set to something like /usr/bin, and we set PATH=, then to run the command, we could type in /usr/bin/(whatever command we want to run). It would just be frustrating because we would need the full path every time.

Give an example of a command that works the same even with an empty `PATH`. Why does this command still work?

One command that still works is pwd. This command works because it is a builtin command/function. It is directly executed on the shell itself rather than having to be externally executed. Something like ls is loaded and executed by the shell. Pwd is just run directly in the shell.

2 Playing with the shell a bit: Special Variables

Bash has quite a few special variables that can be very useful when writing scripts or while working at the terminal.

What does the variable `$?` do? Give an example where this value is useful

$? will return the status code of the last shell command that was run. So if we had some kind of a bash script, and we didn't know where it wasn't working, we could continually run echo $? until we found a return value that was 1. (0 means successful, 1 means unsuccessful). This could be a useful tool in debugging.

What does the variable `$1` do? Give an example where this value is useful

$1 refers to the first argument that is passed into a shell program from the command line. So for example, if I ran something like ./newscript.sh "eric", then $1 would refer to "eric." This can be useful because if we need to access passed in parameters in the shell program, we can use these positional parameters to do so.

3 Basic Scripting

Recall from lecture that scripting is really just programming, only in a very high-level language. Interestingly, `sh` is probably one of the oldest languages in regular use today.

`make` is a good tool for build systems, but we can actually use some basic scripting to accomplish a lot of the same things. First, write a simple C program that prints "Hello World!". Write a shell script named `build.sh` that performs the following actions:

1. Compile your program
2. Runs your program
3. Verifies that your program outputs exactly the string "Hello World!"
   - There are good utilities that check the `diff`erence of two files. They could be helpful.
4. Prints the string "All tests passed." If the output is correct, or prints "Test failed. Expected output >>Hello World<<, got output >>{the program output}<<".

Copy the output of `cat build.sh` here:

```
#!/bin/bash
# Script that compiles, runs, and compares to see if the run file
# has the correct contents in it.

# Compile the program some random name
g++ -o "compiled" $1

# If there were no compilation errors, run the program
if [[ $? -eq 0 ]]; then
    result=$(./"compiled")
    expected="Hello World!"

    #compare the output of the executable with the expected output
    if [ "$result" = "$expected" ]; then
        echo "All tests passed."
    else
        echo "Test failed. Expected output >>Hello World!<<, got output >>{$result}<<"
    fi
fi
```

4 Controlling your environment

In lecture, we added a directory to our `PATH` so that we could just type `hello` and the Hello World program would run. It would be annoying to update the `PATH` variable every time we open a new terminal. Fortuntately, we can do better.

Describe how you would set up your system to modify your `PATH` automatically every time you open a new terminal (what file would you change and what would you put in it?)

We can change the .zshrc file that's located in the home directory (I'm using zsh, but it'd be bashrc if you're using bash). Inside of here we have a command export PATH="yadayadayada." We can add the directory that we want into this export PATH command, so then future calls to hello would search in the directory we added as well!

Roughly how long did you spend on this assignment? 1.5 hours