Advanced Homework 6

Due: Wednesday, February 21st, 11:59PM (Hard Deadline)

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. **Make sure to check the office hour schedule as the real due date is at the last office hours before the date listed above.** This applies to assignments that need to be gone over with a TA only. **Extra credit is given for early turn-ins of advanced exercises. These details can be found on the website under the advanced homework grading policy.**

1 Automated Background Testing

As projects grow, the number and complexity of test cases grows as well. While it's generally a good idea to run all of your test cases, it can be annoying to sit around and wait for a long test case that tests a part of your program that (you think) you didn't touch.

Scripting to the rescue! The goal is to write a script that runs tests in the background. This script will make sure everything builds correctly, run all of your tests, and, as a bonus, run an external correctness checker that can help find mistakes.

Eventually, we'll invoke it like this: (calling the script with the repository to test as an argument)

```
$ ./run-tests /Users/mterwilliger/repos/eecs280-p1
```

But first, read over everything your script should do:

1. Create a directory in `/tmp` that is unique for this project if one does not already exist.
   - *Hint: The project's current directory is already unique*
2. Assuming the directory you created above is at `$PROJ_TMP_DIR`, run
   - `git clone --local $1 $PROJ_TMP_DIR/$(git describe --always)`
3. Call a second script (`run-tests-bg` or something like that) that runs in the background to execute the remaining tasks. This second script should do all of its work in the clone you just made.
   - *Hint: You will likely need to pass arguments to this secondary script*
4. Call `make` to build the project. You should save all of the regular output to a file named `build.log` and any errors to a file named `build.error`
   - If the build fails, your script should stop and notify that the build failed. See the notify section below.
5. Run every test case for this project
   - Your script should assume that any executable file with "test" in the name is a test case. It should **not** hard-code a list of tests to run
   - Your script should save the regular output and the error output of each test case to unique files.
   - Your script should assume that test cases return 0 when they pass and non-zero otherwise.

6. Generate a nice report when everything is done and alert the user using `notify-send`
    - Try running `notify-send "I am a title" "And I am a body"`
    - Your notification should include in the notification what you think is useful. At a minimum it should include how many test cases passed.

Notice that this script is **generic**. It should work in any repository.

To get started, use the EECS 280 W15 repository you created for Homework 2.

Create a directory to hold the scripts you're working on

- `mkdir ~/scripts`

This is going to be a rather complicated script, so it's a good idea to put it under version control as well.

- `cd ~/scripts`
- `git init`

Inside this directory, create a file named `run-tests` with the following contents. Also be sure to make this script executable.

```
#!/usr/bin/env bash

echo "Hello, I am your script running."

echo "Number of arguments I received: $#"
echo "Argument[0] (the program being run): $0"
echo "Argument[1]: $1" # This is empty if no arguments were passed
echo "Argument[2]: $2" # This is empty if only one argument was passed

# Notice what directory this script is executed from. This is important
# when you try to call your helper script
echo "$(pwd)"

sleep 10s && echo "It is annoying to wait for long commands to finish"

sleep 10s & echo "We can put them in the background so we don't have to wait"

notify-send "Test Message" "Blocking part of the script finished"

echo "Notice that the second sleep is still running, we can tell by: $(pidof sleep)"
```

Don't forget to add and commit the starter code

- `git add run-tests`
- `git commit`

Now, let's try it out with the repository you created in Homework 4:

- `~/scripts/run-tests ~/eecs280-w15   # Or wherever you put this`

Some final tips

- Build up your solution in parts! Get step 1 working then commit it. Then onto step 2.
- Steps 1 and 5 are probably the most challenging.
- You, of course, do not need to actually implement project 2 from EECS 280. I recommend something like this:

```
$ git log --oneline -n1 -p
dfaeb04 Force filter_test to always succeed
diff --git a/filter_test.cpp b/filter_test.cpp
index eebeb87..fa1bc1e 100644
--- a/filter_test.cpp
+++ b/filter_test.cpp
@@ -20,6 +20,7 @@ bool isPrime(int x)

 int main()
 {
+    return 0;
     int numbers[] = { 3, 20, 46, 43, 9, 17, 103, 102 };
     const int numSize = sizeof(numbers) / sizeof(int);
     int primes[] = { 3, 43, 17, 103 };
```

Submission checkoff

- ☐ Explain what the clone command from step 2 does.
- ☐ Show off your script working
  - ☐ With all passing test cases
  - ☐ With some failing test cases
  - ☐ Show how to find and examine the output from a failing test case
- ☐ Show that your script is generic by running it with different git repositories that contain tests
  - If you aren't in a programming EECS class, grab any other 280 project.

[Optional Section] Automating Automated Background Testing

Surprise! The way you wrote your script makes it very easy to install it as a git `post-commit` hook. However, one more change will need to be made to the script. The `post-commit` hook doesn't pass in the repository path as the argument to the script. Instead, you will need to check in your script if the repository path is given to the script. Based on that information you know whether the script is run by git or by a user (since a user would pass in the repository path). If it's run by git, you'll need to know what directory the script is run from. Some info from the git docs contains this information:

> Before Git invokes a hook, it changes its working directory to either `$GIT_DIR` in a bare repository or **the root of the working tree in a non-bare repository.**

Emphasis mine.

- Hint: `pwd` might come in handy here

Once those changes have been made, try:

- `cd ~/eecs280-w15/p2/.git/hooks  # Or wherever you put this`
- `ln -s ~/scripts/run-tests ./post-commit`

The command `ln -s` creates a symbolic link, or symlink. Recall symlinks are just like pointers. This design lets us install the same commit hook into multiple repositories just by pointing to it. If we ever update or make improvements to the hook, all of the repositories that use it will automatically get upgraded.

All right, we are finally set up. Make a new commit and test out the new hook!