Homework 4
Git

Assigned: Friday, January 29, 11:00AM

Due: Friday, February 5, 11:00AM (Hard Deadline)

Submission Instructions

Submit this assignment on Gradescope. You must submit every page of this PDF. We recommend using the free online tool PDFescape to edit and fill out this PDF. You may also print, handwrite, and scan this assignment.

Even if there are not questions to answer on the first page, please still submit every page of this PDF.

There may multiple answers for each question. If you are unsure, state your assumptions and your reasoning for why you think your answer makes sense.

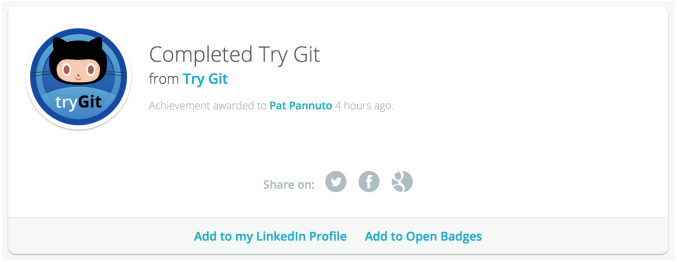
1 Git's Chicken and Egg Problem

Git has a (well-deserved) reputation as a tool with a high learning curve. There are several reasons for this. One is that git has a bad user interface – the commands and flags you use to get things done are frustratingly inconsistent and simply must be learned over time. The bigger reason, however, is that using git requires having an understanding of how git works and learning how git works requires having some experience using git. This circular dependency makes it hard to get started as there is no good way to break in. The best we can do is try to hold your hand as you get started.

Code Academy and GitHub teamed up to build a nice, online git beginners tool. As you work through the exercises, pay attention to the explanations of each step at the top of the page, and the "Advice" section at the bottom/bottom-right of the page. The text introduces several new pieces of git-specific language.

Complete the online course at `https://try.github.io`. Grab a screenshot of your badge and copy the URL here. (You will need to sign in or create an account get your badge)

My badge and URL as an example:



https://www.codeschool.com/users/2276474/badges/121

Your badge here

Some other tutorials / resources

There is a *lot* written on the Internet about how to learn git. Some highlights:

- For a deeper dive into git from the basics to the very advanced, check out Atlassian's tutorials (Atlassian is the company behind BitBucket and JIRA).
- For more visual learners, check out this interactive visual learning game – caveat this game simplifies some things, but I think it's still a useful tool to learn from. There's even some low-hanging fruit lying around to improve this tool.

2 Using git in your own projects

This question sets up the Winter 2015 EECS 280 P2 as an example. Don't use it for this term!!

0. Install git: `sudo apt-get install git`

Using version control well is a habit learned over time. As a first step, for the rest of the term, the **first** thing you should do whenever an EECS class assigns you a project is:

1. Create a repository

```
> mkdir -p eecs280-w15/project2
> cd $_
> git init
```

2. Grab a copy of the spec

```
> wget 'https://drive.google.com/uc?id=0B4qlH840ZwikRmQtdkRIeTZjODA&export=download' -O spec.pdf
```

3. Grab any starter code

```
> wget 'https://drive.google.com/uc?id=0B4qlH840ZwikbkZLS3Z5YTVSeW8&export=download' -O eecs280-w15-p2.tgz
> tar -xf eecs280-w15-p2.tgz
> rm $_
```

4. Add it all to git and commit it

```
> git add *
> git status # Check to make sure everything looks right
> git commit
```

Do this **before** even beginning to read the spec. Make it a habit.

If you try typing `make`, you'll find that this starter code does not compile. The spec has a great tip at the bottom of page 11 that suggests adding stub functions.

There are 15 functions we need to stub out. That could be a pain, but fortunately the header file has already done most of the work. We'll drill into this command next week, but for today, simply run:

```
> echo -e '#include <cassert>\n#include "p2.h"\n' > p2.cpp
> grep ';' p2.h | grep -v ' \*' >> p2.cpp
```

One of the optional things to learn about editors from last week's homework was macros. In case you missed it, let's demonstrate their use here:
Note: I use vim, but emacs also supports macros with `C-x (`, `C-x )`, and `C-x e` to record, finish, and execute.

```
> vim p2.cpp
# press 'jjj'    so that your cursor is on the i of int sum(...
# press 'qq'     to begin recording a macro into the vim register q
# press 'f;'     jump to the ';' character
# press 's'      remove character under the cursor and enter insert mode
# insert the needed text:
#   '<space>{<enter>assert(false);<enter>}<enter><escape>'
# press 'j'      so that your cursor is on the i of int product(...
# press 'q'      to finish recording the macro
# press '16@q'   14 times play the commands stored in register q
# save and quit vim
```

Now try `make`. Cool, a building project in 60 seconds or less.

Commit the changes.

You ran `git status` before committing those changes, right? (Because it's *always* a good idea to run `git status` *all* the time). You probably noticed that annoying "Untracked Files" section that lists all of the compiled output the Makefile made. It doesn't really make sense to track compiled output in a version control system (it is hard to track changes on a binary file such as an executable).[1] Instead, we would prefer if git would ignore the built output.

Commit a change so that built output is ignored. When you are done, `git status` should print

```
On branch master
nothing to commit, working directory clean
```

[1] Some binary files are good to check in though, such as the project specification, which will rarely or never change.

Once you have completed all of the steps on the previous page, copy the output of:

> **`git log -n2 -p | head -n 40`**

The only real question here required you to go learn about the `.gitignore` file. This file is a convenience that tells git about the files that you explicitly do not want to track, such as built output. The purpose is the prevent them from showing up as "Untracked Files" all of the time in `git status`.

While you can list files explicitly one-by-one, it's often more useful to use wildcards (the `*` operator). My solution does this to automatically ignore any file ending in "`_text`".

GitHub maintains a list of template gitignore files that can be a good place to start from.

```
ppannuto@ppannuto-c4cs-vm:~/hw4$ git log -n2 -p | head -n 40
commit 8f614fbf43150236055000b53a27ba8748f8438f
Author: Pat Pannuto <ppannuto@umich.edu>
Date:   Thu Feb 4 23:34:18 2016 -0500

    Add gitignore

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..eaac059
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,2 @@
+p2-tests
+*_test

commit 7cf44db2e47b13a402d10d404448a811d84f0c78
Author: Pat Pannuto <ppannuto@umich.edu>
Date:   Thu Feb 4 23:33:21 2016 -0500

    Stubs

diff --git a/p2.cpp b/p2.cpp
new file mode 100644
index 0000000..55cc9eb
--- /dev/null
+++ b/p2.cpp
@@ -0,0 +1,54 @@
+#include <cassert>
+#include "p2.h"
+
+int sum(list_t list) {
+       assert(false);
+}
+int product(list_t list) {
+       assert(false);
+}
+int accumulate(list_t list, int (*fn)(int, int), int identity) {
+       assert(false);
+}
+list_t reverse(list_t list) {
```

* * *

We will do more with version control again after spring break. It is important that you have at least one project that you have been using git with by then.

* * *

3 Pulling some pieces together

Try the following:

```
# The /tmp directory holds temporary files. It's a great place to do some
# quick tests or experiments, however it is automatically emptied every time
# the machine reboots. This means if your machine crashes, you *will* lose
# everything in /tmp.
#
# Don't ever put anything you care about in /tmp !!

> mkdir /tmp/throwaway
> cd /tmp/throwaway
> git init
> echo hello > world
> git add world
> git commit
```

Git automatically opens a text editor for you to enter a commit message.

What text editor did git open for you? _____

> This will change depending on whether you've installed a `vim` package as part of a previous homework. The `vim` packages will set up `vim` to be the default system editor, in which case this will open `vim`. If you haven't installed `vim`, it will open `nano`.

Git is not the only program that will open an editor. If you don't like the default choice, we can change it using the environment variable `EDITOR`. Try the following:

```
> echo "it's a small" > world
> git add world
> export EDITOR="emacs -nw"
> git commit
```

What text editor did git open for you? _____

What does the `-nw` flag do? (Hint: Try it without `-nw`)

> It will now open emacs as the editor for commit messages.
>
> The `-nw`, or "`-no-window-system`", option tells `Emacs` not to create a new window, rather to open in the existing terminal, like the default behavoir of `vim`.
>
> You can also go the other direction and ask `vim` to open in its own window like `Emacs` by running `gvim` instead of `vim`.

Perhaps, however, you do not want to change the editor used for every program, but only for git. We can do that to. Try the following:

```
> echo "on top of the" > world
> GIT_EDITOR=gedit git commit world
```

If you set `EDITOR` and `GIT_EDITOR` to different values, which takes priority?

> Usually, more specific things take priority over more general things, so `GIT_EDITOR` takes priority here.

Describe how to "permanently" change the default `EDITOR` for your system to emacs (Hint: What files did Q1 from week 2 read on startup?)

> A couple choices here:
>
> - Add "`export EDITOR=emacs`" to `~/.bash_profile` or `~/.bashrc`
> - You could also configure git itself: `git config --global core.editor "emacs"`