Homework 3
Git

EECS 201 Winter 2020

Submission Instructions

This homework will be submitted as a repository on the UMich GitLab server. This will become evident as you work through the assignment.

Preface

Git has an interesting conundrum where to use Git you need to have some understanding of Git, but to understand Git you need to have some experience using Git. While this class' goal is to help you become more independent problem solvers, this circular dependency may make Git hard to get into, so for this homework we'll give a guided introduction.

Some other tutorials/resouces you can look at besides the official documentation and *Pro Git*:

- Atlassian's tutorials
  Atlassian is the company that made BitBucket, JIRA, and Confluence

- Interactive branching game

- A humorous quick-reference guide

- A more humorous quick-reference guide (language warning)

In this introduction we will be taking an existing codebase and turning it into a Git repository. The process of doing this is an empty directory is exactly the same: instead of an empty directory we have untracked files that are already provided.

1   GITing started

You are in charge of migrating a small command line calculator program from an archaic, proprietary version control system whose company became defunct in the 90s to Git as well as handling the remaining issues in the code.
First, we'll need to acquire the code:

```
$ wget https://www.eecs.umich.edu/courses/eecs201/files/assignments/hw3-calculator.tar.gz
$ # e(X)tract (Z)ipped (F)ile
$ tar xzf hw3-calculator.tar.gz
```

1. Download and extract the code. `cd` into the extracted directory.

2. Let's play around a bit with the application. Run `make` to build the application. Familiarize yourself with the source code in `src` and `inc`.

3. Try running `$ ./calculate 3 x 3`

4. Let's git started. Initialize a Git repository with `git init`.

5. Ignoring my advice in lecture let's blindly add all the files in the current directory with `git add .`

6. Run `git status` to see what's currently in the Index (get into the habit of running `git status`; it'll tell you a lot about what's going on).

7. Notice how we have the final compiled binary `calculate` and intermediate object code files in `obj` in the Index. In general, we don't want to version the intermediate and final build outputs as they're the products of the code that we are versioning; no reason to version a thing twice. Other things that we don't tend to version include development environment specific things, like logs, other output files, or core dumps that your application produces during runtime, and developer system dependent things like editor swap files and weird filesystem helper files like `.DS_STORE`.

8. Use `git reset` to unstage the `calculate` binary and files under `obj`. If your OS or filesystem creates any other junk specific to it (like `.DS_STORE` on macOS), unstage those as well.

9. Use `git commit` to make this initial check-in/commit of the codebase. Be sure to make a commit message that follows best practices as mentioned in class.

10. Run `$ git status` for good measure. Isn't seeing `calculate` and `obj/` as untracked files annoying? Figure out a way to get *Git* to locally *ignore* the intermediate and final build outputs, staging the necessary file(s) and making a commit. If your OS, filesystem (or text editor) creates files like `.DS_STORE` or `*.swp` files, you may want to look into a way to ignore such files `globally`.

11. Run `$ make clean` to delete the intermediate and final build outputs.

12. Create a README file that explains how to build and clean the application. Stage and commit it.

13. `rm` the `Makefile` and delete a character from the README.

14. Recover the deleted `Makefile` undo the changes to the README using `git checkout` or `git restore`.

Hooray! We've migrated the application to Git. Now we have to move onto fixing our problems.

2   GITing around

1. Take note of the comments that mention an issue in the source code.

2. From here, we are going to make two `topic branches` that deal with each of the issues. Use `git branch` to create branches `issue149` and `issue221` that correspond to the issues mentioned in the comments.

3. Switch to one of topic branches using `git checkout` or `git switch` and perform its corresponding fixes.

    (a) Make sure to delete the comment mentioning the issue in the code, as you have now dealt with it.
    (b) For Issue 221, when handling the divide by zero return status, returning the proper code and print "Divide by 0" with a newline to `stderr`.

4. Use `git diff` to see how your files have changed.

5. Use `git add -u` to add files that you have modified. If you have staged files that are unrelated to this issue, be sure to unstage them as this branch is focused on fixing the appropriate issue.

6. Commit the fix, making sure to have a properly formatted and descriptive title and body for the commit message as well as including a reference to the issue number in the **body**.

7. Repeat steps 3 to 6 for the other branch.

8. Switch to the `master` branch.

9. Now we are going to bring the commits from the topic branches over to the `master` branch.

10. Use `git merge` to bring over the commits from one of the topic branches. Note that this topic branch and `master` **have not** diverged, making this process painless. This is known as `fast-forwarding`, where a branch just moves its pointer up.

11. Use `git merge` to bring over the commits from the remaining topic branch. Note that this topic and `master` **have** diverged with the topic branch being a commit or two (or more) behind `master`. This will necessitate a special *merge commit* that gets automatically generated (which you don't have to reword for this assignment). If the merge has to stop due to not being automatically resolved, use `git status` to see where the merge conflict is occurring and modify the files to get them into working order, then complete the merge.

12. Run `$ git show` to see what this latest commit entails.

13. Run `$ git show HEAD^1` or `$ git show HEAD^` to see the first parent of this merge commit.

14. Run `$ git show HEAD^2` to see the second parent of this merge commit.

15. Use `git log` to show you the list of commits.

Now that your job is finished, let's push it to a remote repository.

3 You're going surfing on the internet!

1. Log into https://gitlab.umich.edu and set up your UMich GitLab account. **(Don't mistake this for the EECS department's GitLab server gitlab.eecs.umich.edu!)**

2. Set up SSH with your UMich Gitlab account (if you haven't already). You can find this under your account settings > SSH Keys, and the site has a guide on how to set it up. This allows you to painlessly clone/pull/push with the UMich Gitlab server. (The counterpart to SSH is HTTPS, but SSH is seriously easier to work with on a regular basis).

3. Create a completely blank, new **private** "Project" (i.e. remote repository) with exactly the name: `eecs201-hw3`

4. Let's now set up this UMich Gitlab project as the remote for your local repository.

5. Add it as the `origin`: `$ git remote add origin git@gitlab.umich.edu:<your uniqname>/eecs201-hw3.git`. This uses `git remote` to add a remote named `origin` with a specified SSH URL.

6. Now let's push the all commits from all the branches to the remote: `$ git push -u origin -all`. `-u` sets the upstream tracking information for the local branches, allowing them to push/pull commits from the remote branches. `origin` is the remote that we are referring to. `--all` pushes all of the branches. You could instead, say, push only `master`.

7. In the UMich GitLab project's Settings>Members add the course staff `brng` and `computer` as Reporters. As part of the grading process for this homework we will be looking at your repositories on the UMich Gitlab.

8. Back in your local repository, make sure you are in `master`.

9. Create a file called `report.txt`.

10. On the first line of the file, put down an estimate of how long you took to do this assignment in minutes as an integer (e.g. "37", "84": just numbers, no letters).

11. On the second line and onwards of the file, put down what you learned (if anything) by doing this assignment. If you already knew how to do all of this, just put "N/A".

12. Stage and commit this `report.txt`.

13. Use `git push` to push this commit to your UMich GitLab repository.