Weld Lab git/GitHub Primer

Peter Dotti

1   Introduction

This document is intended to establish basic git and GitHub practices in the Weld Lab and to be a simplified introduction to using git and GitHub. The hope is that implementing these practices will enable the following:

1. Editing code without fear of disrupting a working version

2. Easily update versions of code once a testing version of the code is established

3. Inherent record keeping and version recording as automatically carried out by the git software

4. Backup versions (current and historical) of code as saved on both GitHub and the Citadel server

5. A unified procedure so that accessing and using other lab member code is easy

While there is desktop software for GitHub (and, presumably, git more generally), I found it not to be sufficiently intuitive, robust, or easy to justify using it over command line inputs, so this document will focus on use of command line git.

Hopefully it will be easy and painless to do these things. Please email / slack message me (Peter) if you have suggestions to improve this document in the future. Additionally, I hope to use git on this document itself for easy editing.

2   An Important Reference

If you find that this primer is lacking some piece of information that you would like, I strongly recommend using the online "git Book" from https://git-scm.com/book/en/v2. I found it to be very well written and well organized, although it will certainly have more details than you need (although, you will likely not need most details past the fourth chapter, unless you wish to do more advanced things).

3   Using git on your Computer

Git revolves around the use of "repositories." A repository is just some directory (folder) on your computer with a hidden git file in it that designates it as a repository and allows you to set and keep track of files.

You are able to change the files in the repository however you'd like, using whatever code editor or program you'd like, but you will regularly want to make a "`commit`" with git so that the version

of the repository is saved. Additionally, you will want to occasional use the "`push`" command to use the current version of the repository on your computer to update some remotely saved version of the repository (i.e., you will use `push` to update a stored version of the code on GitHub and the Citadel)

From here on, I will assume that the reader has some command line tool with git installed on it (instructions on how to do that here: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git). I will also assume that the reader knows how to change directories and get directory path names in the command line.

3.1 Making and Using a git Repository on your Computer

Once you've navigated to the directory that you want to initialize as a repository, you run the command

```
git init
```

At this point, you can fill the directory as usual (assuming it didn't already have code in it). If you want something to test basic git functionality, I would recommend saving a text file in the directory and editing it.

At this point, you can use the command

```
git status
```

And it will show something like

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
  <A list of the files in the directory>
```

The general meaning of this is that git knows that you have files in the directory that you might want to use as a repository, but they are currently "untracked" and git is waiting for you to indicate which files you want it to keep track of edits for (and moreover, which files it should copy over to an external server such as GitHub or the Citadel when you ask it to do so).

To indicate which of the files in the list you want it to track, use the command:

```
git add <filename>
```

for each of the files you'd like. If you want it to track all of the files, you can conveniently use

```
git add --all
```

There are fancier patterns to use. I believe they generally follow the UNIX wildcard characters. Resources exist online to learn them, such as https://geek-university.com/linux/wildcard/. I'll give one example:

```
git add *.m
```

Git will look for any file of the form <SOMETHING>.m, i.e., it will add any .m matlab file to the repository.

You can now run

```
git status
```

again, and you should now see an indication that some files have been added. The jargon is that these files are "staged", and they are ready to be "commited."

Now it is time to use the `commit` command to tell git that you want it to mark this as a point in history that you would like remembered. In other words, the `commit` command saves version of the current repository state.

Use the command

```
git commit -m "<Message describing this commit>"
```

The message you enter should describe this version or why it is different from the last commit.

And that's about it for a repository on your computer! You can now go along and make edits to the files in your repository. Then you make commits whenever you see fit.

The one last note I should make is that whenever a file is edited and you want to make a new commit, you have to once again use the `git add <filename>` command to once again stage the modified files (as if you were adding a new file again), before using the `commit` command. However, it is generally more convenient to skip the `add` step and use

```
git commit -am "<Message describing this commit>"
```

If you missed it, I added an "a" between the - and m. That "a" tells git to stage all of the modified files (but *not* new or untracked files) and make a commit.

3.2   Checking Status and Logs for a Repository

In general, you can get helpful information about the status and history of a repository with the `git status` and `git log` commands.

We have already seen use of `git status` above. It will typically tell you what files have been modified or what new files have been added to the repository since the last commit. It will also tell you whether or not the current commit has been used to update the remote repositories (e.g., GitHub or the Citadel).

`git log` will tell you about the version history of the repository. Specifically, it will list out the previous commits in reverse chronological order. Moreover, it will tell you if you "branched off" other versions and merged them later on in the history (more info later).

I recommend `git log --graph --decorate` which makes a sort of ascii art representation of the log history.

3.3 Looking at an Old Version

Now, suppose you think there was some code in a previous version that was lost or abandoned and needs resurrection. We will now give a simple way to temporarily restore to that old version and view the file. There are other methods, and I am frankly not sure if the `reset` command is more appropriate in most cases, but this seems the least invasive method of what I've learned in my bit of prep.

First, you will want to get the identifying number for the old commit that you want to see. To get it, run `git log`. Each commit will have a number (in hexadecimal) after it. It will look like `commit 9a52cbc45aa6f5c6064c654ef7951081930274fb` for example.

You can now use the command

```
git checkout 9a52cbc
```

(you only need the first 7 digits). It will give some statement about going into 'detached HEAD' state (which sounds worse than it is) along with some helpful advice.

Your files will now be in the state of the old commit corresponding to the number you gave it. You can open the code from the repository directory in the usual way and it will be like you've gone back in time to that old commit. You can even make a new branch with edits from it to save that state and use it as something to work on separate from the newest version.

To go back to the newest version, run

```
git checkout <branchName>
```

`<branchName>` will almost certainly be `master`, unless you have changed it (or want to checkout a different branch). To see what named branches you have, you can run

```
git branch
```

and it will return a list of branches (more on branches in the next section).

3.4 Branches

I would like to keep a discussion of branches as brief as possible here. A fuller introduction is at https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell. However, I believe it is necessary to know what a branch is to use remote repositories on GitHub and the Citadel, and this is one of our main goals.

As we saw before, git allows you to restore a repository to the state at a previous commit. A natural question is "Instead of just working with previous and current versions of a code set, is there a way to have multiple different working versions of code that sort of branched off from an earlier version?" The answer is yes, and git calls these different working versions "branches".

These branches can be especially useful if you want to keep a working version of the code as the master branch (git's preferred term for it) while editing in added functionality.

4 Using git with Remote Repositories (e.g. GitHub and the Citadel)

Test