Lab 2: Project groups and GitHub

Due. Tuesday, June 11th, 2019

In this lab, you will form your project teams. Then, you will work with your team to create your GitHub repository.

What To Do

Part 1: Find your team

The first step in a team project is finding team members. When picking team members, you might consider their work schedules, work habits, and experience with various technologies.

Step 1. Form a group of 4, 5, or 6 students. Create a private **Slack** channel and invite all members of your team.

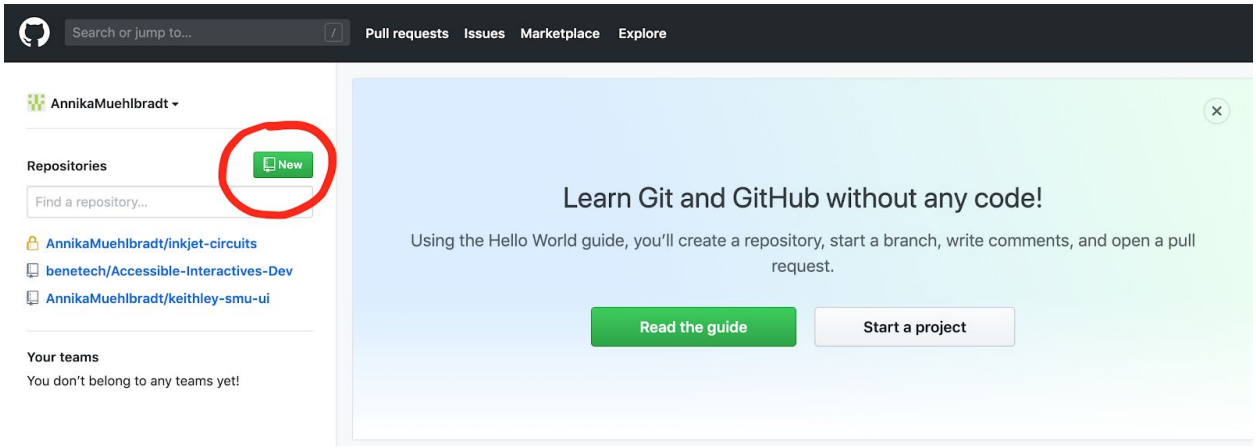
Step 2. Come up with a team name or moniker for your project.

Part 2: Setup GitHub

Now, you will create a repository for your team. You can choose any version control repository hosting service, but GitHub is strongly recommend. Note that the free version of GitHub only allows up three collaborators for private repositories. If you want your repository to be private you can use GitLab.

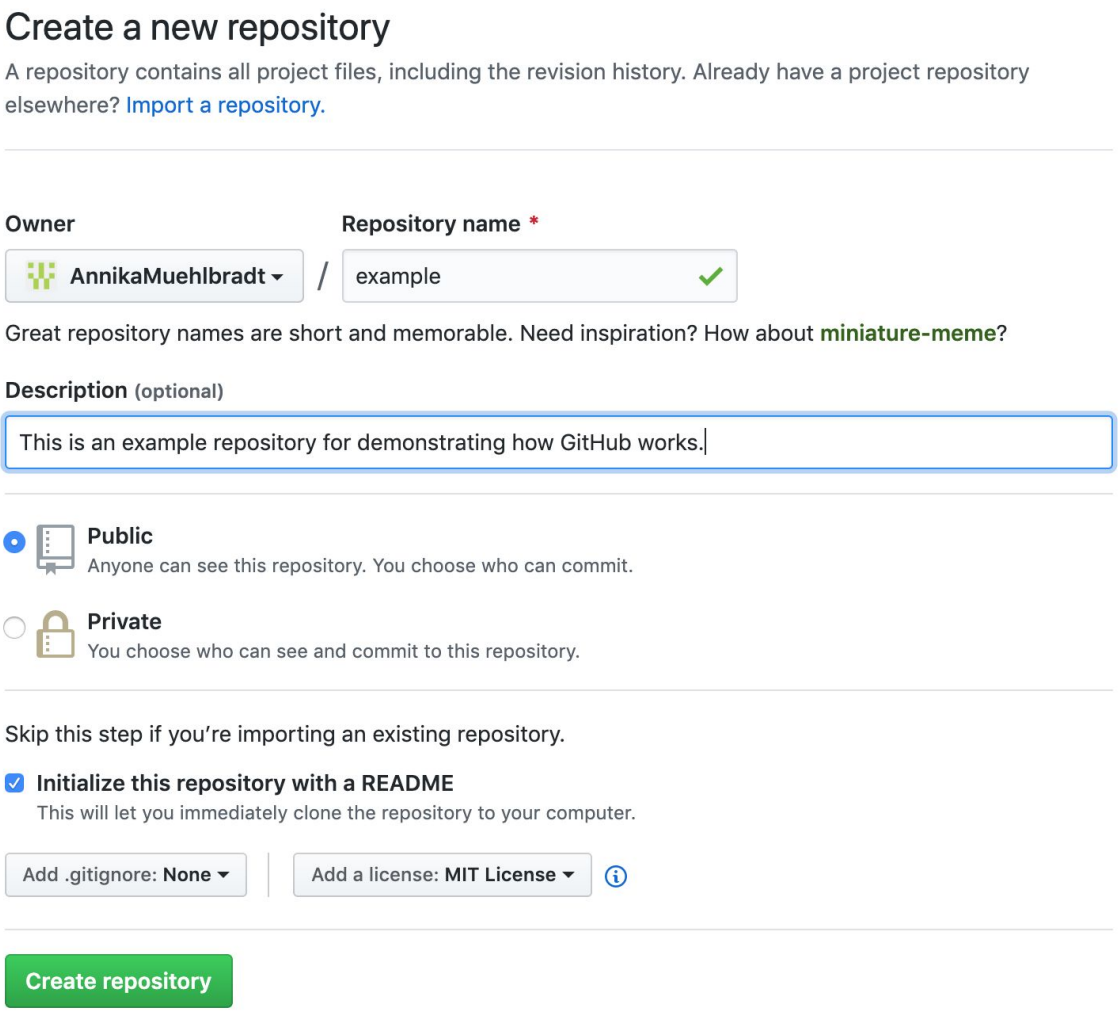
Step 1. Everyone on your team should create an account on GitHub. If you already have an account, skip this step.

Step 2. One member of your team should create a new GitHub repository. Use the green button labeled “New” on the left hand side.

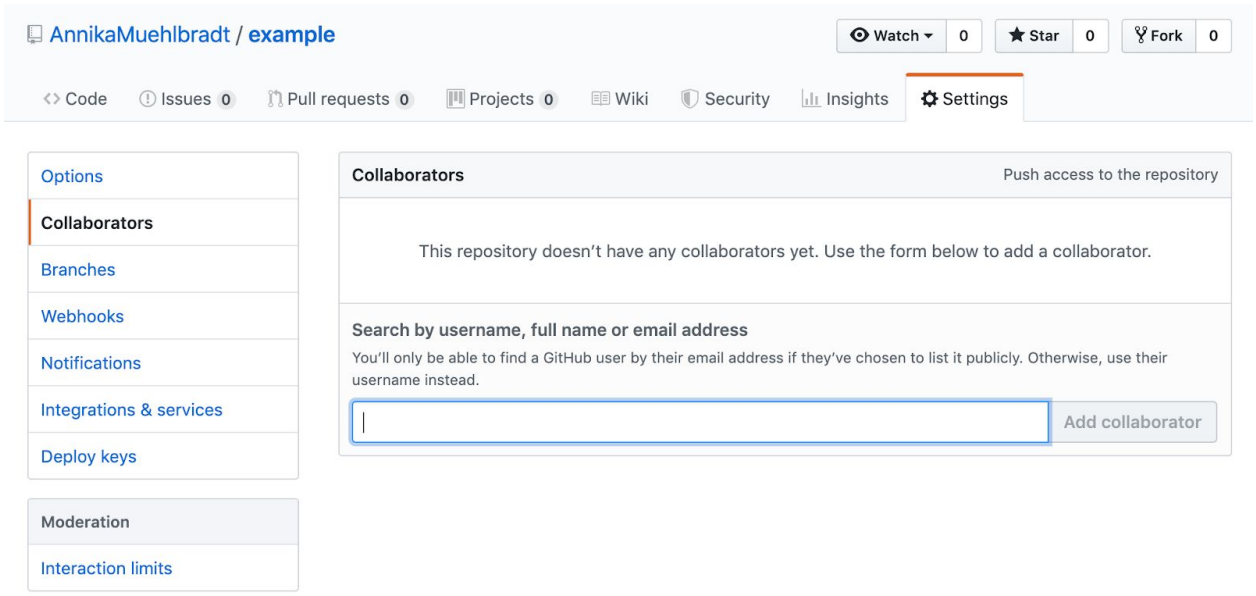


Fill out the following fields:

1. Name your repository. This should be your team name or project moniker.
2. Provide a one sentence description of your project (optional)
3. Make sure your repository is public.
4. Select “Initialize this repository with a README.”
5. Leave “Add .gitignore” as “None”
6. Under “Add a license” select “MIT License”



Step 3. Add all the members of your team to the repo. Go to Settings > Collaborators. If you are using GitLab and you created a private repo, make sure to add your instructor as a team member to your repo.



Step 4. Everyone on your team should setup Git on their computer.

Mac users may already have Git installed. In your terminal, run the command "git --version":

```
$ git --version
git version 2.20.1 (Apple Git-117)
```

If Git is not installed you can run the command "xcode-select --install" to install the MacOs command line tools, including Git:

```
$ xcode-select --install
```

A popup will appear. Select "Install"

Windows users can download Git here.

Linux users can install Git by running the following command in the terminal:

```
$ sudo apt-get install git
```

Step 5. Configure your Git username and email address to match your GitHub account. In your terminal, enter the following commands:

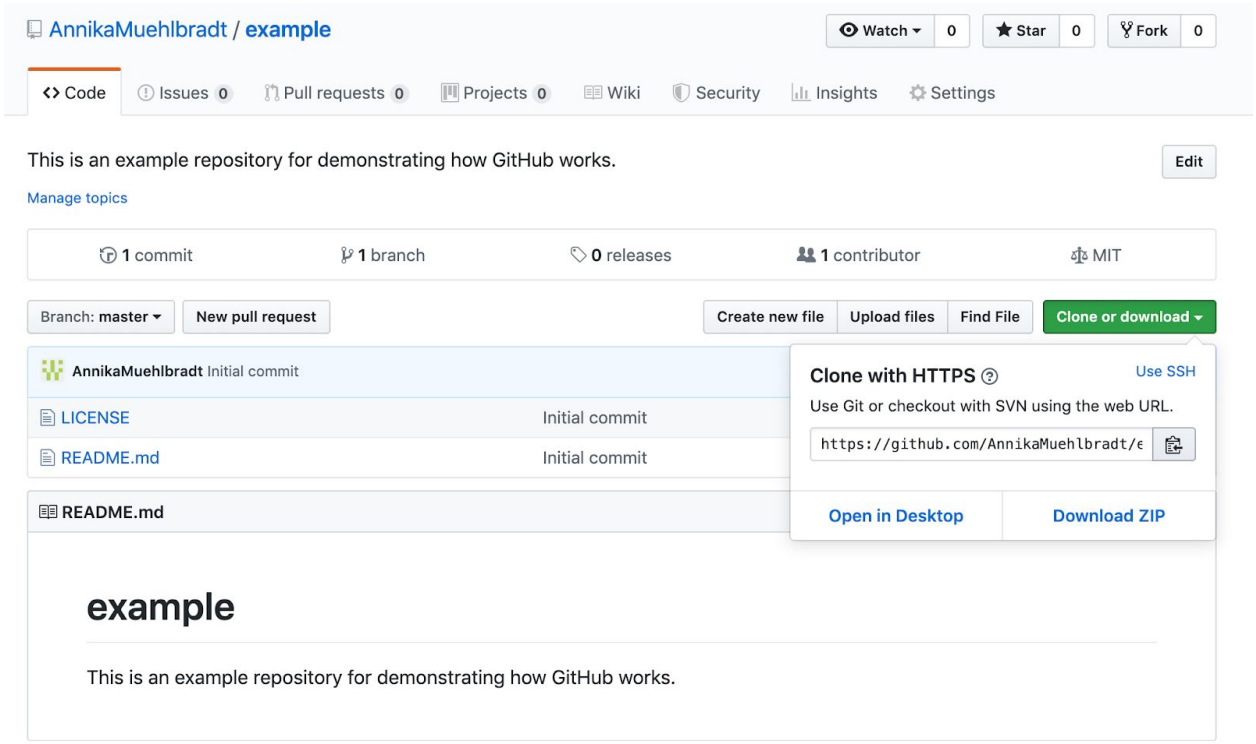
```
$ git config --global user.name "Your GitHub username"
$ git config --global user.email "Your GitHub account email address"
```

Verify that you have entered your username and email correctly with the following commands:

```
$ git config user.name
$ git config user.email
```

Step 6. You will now clone your GitHub repository to create a local repository on your computer. Everyone on your team should do this step. It is highly recommended that you create a project folder and clone the repo into this folder.

First, get your repos URL from GitHub. Click the green “Clone or download” button and copy the URL.



Inside your project folder on your computer, run the following command:

```
$ git clone <URL>
```

Again, the clone command clones the remote repository and places it on your system as the local repository. It also sets the **shorthand name** for the remote repository to **“origin”**. The shorthand origin can now be used in place of the remote repository’s URL.

Step 7. Verify that everything works on your system. Enter the directory in which git is tracking the files and make sure the LICENSE and README.md files and the .git directory are there. Everyone on your team should run the following commands on their system:

```
$ cd <your repository name>
$ ls -a
```

Now, everyone on your team should create a file, add it to staging, commit it, and push the changes to the repository. Then, everyone on your team should update their local copy of the repository. Run the following commands.

```
$ touch <your git username>.txt
$ git add .
$ git commit -m "Enter a commit message here"
$ git push origin master
```

Note, how the git push command works.

```
$ git push <shorthand name of remote repo we want to push changes to>
<name of the branch in the remote repo we want to push changes to>
```

By default, the remote repo is named "origin" and the changes are being appended to the branch "master". So at the moment these two commands do the same thing:

```
$ git push origin master
$ git push
```

In fact, if you run the second command, git will tell you that everything is already up-to-date. Try it out. Run git push again like so:

```
$ git push
```

To update your local copy run the following command:

```
$ git pull origin
```

Similarly to the push command, git pull works like this:

```
$ git pull <shorthand name of remote repo we want to pull from>
```

Since "origin" is our default we can omit it from the command and simply type:

```
$ git pull
```

(Optional) Step 8. The name “origin” can be confusing so you might consider changing the shorthand name for your remote repository. Before making any changes, you should always check the status of things. Run the following command to view your existing remote repositories:

```
$ git remote -v
```

Note that it lists the remote repository twice, once for fetch and once for pull. Now, change the shorthand name “origin” to something else. Remember, you are only changing the shorthand name on your system and this change won’t affect your team members.

```
$ git remote rename origin <new name>
```

Now, check the shorthand name of your remote repository:

```
$ git remote -v
```

Part 3: Practice Git commands

You and your team will now practice creating branches, checking out branches, and merging files.

Step 1. One member of your team should create a new branch, add a file to the branch, and commit it to the remote repo.

First, check which branch you are on.

```
$ git branch
```

You should be on the master branch. Now, create a new branch:

```
$ git branch <name of your new branch>
```

You need to checkout the branch in order to make changes to it. By checking out a branch, you are simply switching your working branch. That is, you are pointing HEAD at the checked out branch.

```
$ git checkout <name of your new branch>
```

Verify that you are now pointing at your new branch:

```
$ git branch
```

Create a new file, stage it, commit it, and push it to the remote repo:

```
$ touch <filename>
$ git add .
$ git commit -m "Your commit message here"
$ git push
```

You probably got one of two errors. If you changed the shorthand of your remote repo (Step 8) you likely got this:

```
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote
repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

The git push command uses "origin" as a default push destination. So now we have to tell git which remote repo to push to like so:

```
$ git push <shorthand name of remote repo>
```

If the shorthand name for your remote repo is "origin" you probably still got this error:

```
fatal: The current branch new_branch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream remote new_branch
```

This means that git does not know where to attach the new branch in the remote repository. We can solve this problem in two ways. We can tell git which branch to use like this:

```
$ git push origin <name of your new branch>
```

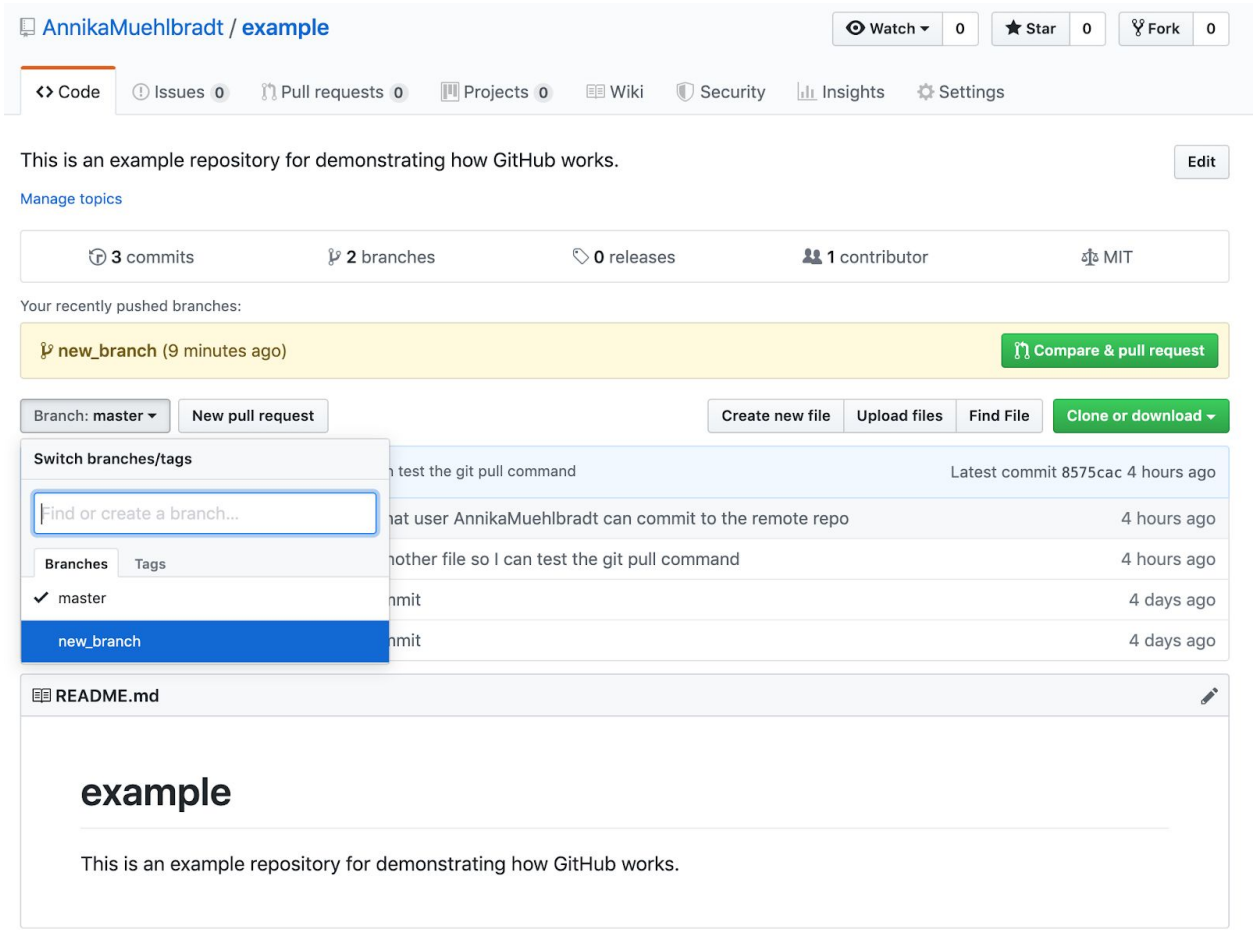
Note that this will create a new branch in the remote repository if the branch you specify does not exist.

Alternatively, we can set git's push default to push the branch to a branch with the same name in the remote repo. Then push the changes. To do this, you can run this command (you can still run this command even if you ran the previous command "git push origin <Name of your branch>":

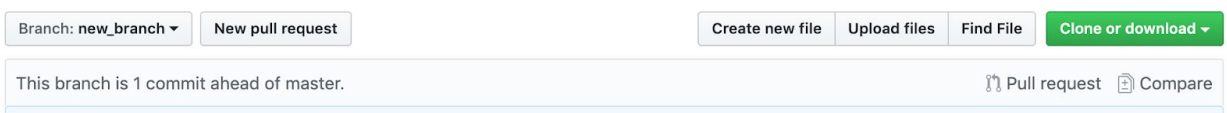
```
$ git config --global push.default current
$ git push
```

Again, if a branch with the same name doesn't exist in the remote repo, git will create one.

Now, go check your GitHub repo to see the new branch.



Note, GitHub gives us some useful information about the branch: "This branch is 1 commit ahead of master."



Step 2. All the team members (except the person who created the new branch) should now update their local repo to see this change. First, let's take a look at our local branches local vs. our remote branches with these commands:

```
$ git branch
$ git branch -r
```

Then, let's fetch all new changes from the remote repo using this command:

```
$ git fetch <shorthand name of remote repo>
```

Now, you need to create a local branch that tracks the remote branch. The following command will create a local branch named <name of your remote branch> that track the remote branch <shorthand name for remote repo>/<name of your new branch>:

```
$ git checkout --track <shorthand name of remote repo>/<name of
remote branch>
```

Note, the option "--track" is a shorthand for

```
$ git checkout -b <name of new local branch> <shorthand name of
remote repo>/<name of remote branch>
```

You can use the above command if you want to name the local branch something other than the name of the remote branch (e.g., "git checkout -b new_feature origin/big_feature" will create a local branch named "new_feature" that tracks the remote branch "big_feature"). This is not advised as you'll lose track of which local branch is tracking which remote branch.

Now that you have checked out the branch, when you push your changes the remote branch will be updated.

Everyone on your team can now add a file to this branch, commit it, and push the changes. Make sure each of your file names are different. Here are the commands:

```
$ touch <filename>
$ git add .
$ git commit -m "Your commit message here"
$ git push <shorthand name of remote repo>
```

Step 3. One of your members will now merge your new branch and your master branch. Again, first update your local repository. You could just use the git pull command. Remember, the git pull command downloads the changes from the remote repo and automatically merges these with your local repo. This is not necessarily a safe operation if you are unsure of what new changes are in the remote repo. Instead, use git fetch to first download the new changes like this:

```
$ git fetch <shorthand name of remote repo>
```

Then, you can look at the new changes before committing them to the local repo. Use the following command:

```
$ git log <shorthand name of remote repo>/<name of remote branch>
^<name of local branch>
```

This will output all of the new stuff that has been added to the remote branch since your last update. You will see all of your team members commits. Now merge these changes with your local branch. To ensure you are merging the changes of the remote branch with the correct local branch, check out the local branch and then merge:

```
$ git checkout <name of local branch>
$ git merge <shorthand name of remote repo>/<name of remote branch>
```

Now, you will merge your branch with your master branch. Again, first make sure everything is up-to-date. Checkout your master branch:

```
$ git checkout master
```

Git will tell you that your master branch is up-to-date. If it is not, repeat the steps above for your master branch.

You are almost ready to merge. You might consider doing one last check of the state of your repo. You can the following command to get a visualization of your repo:

```
$ git log --graph --simplify-by-decoration --all
```

Now let's merge. Run the following command to merge your branch into your master branch:

```
$ git merge <name of branch>
```

Note, git merge always merges the named branch into the checked out branch. Run the git log command again to see that you have successfully merged the two branches:

```
$ git log --graph --simplify-by-decoration --all
```

Don’t forget to push these changes to the remote repository for everyone else to see. Run the following command:

```
$ git push <shorthand name of remote repo>
```

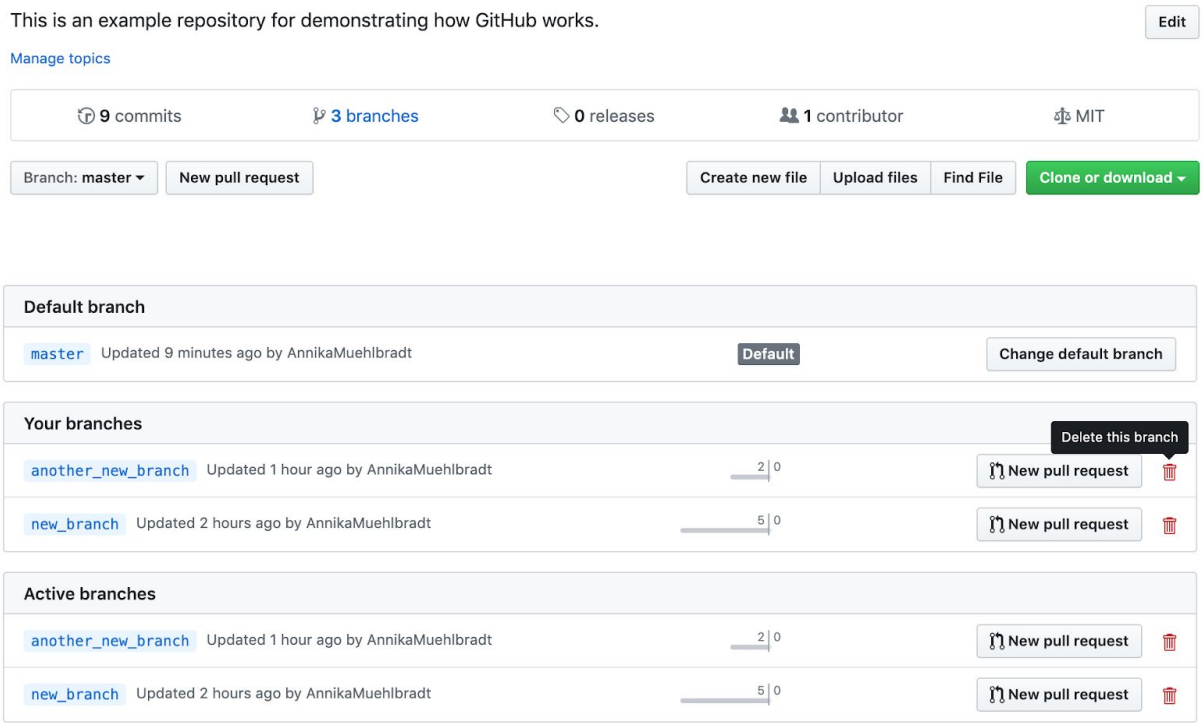
Go check GitHub. When you look at your branch, it should now say: “This branch is even with master.”

Step 4. Take a screenshot of your GitHub repository. Then, run the git log command and take a screenshot of the output:

```
$ git log
```

Every member of your group should submit these two images to Canvas.

Step 5. Clean up your repository. There are several ways to delete branches in the remote repository, you can read about these online. The important thing is that when you are done cleaning up, your remote repository and your local repository should be synced. One easy way to clean up, is to use the GitHub interface. Click on the branches tab and use the trashcan icon to delete the branch:



Deleting the branches on GitHub will not automatically delete the branches from git. We have to remote the stale branches ourselves. To demonstrate this run the git branch command:

```
$ git branch
```

Check which branches are already merged in master and can be removed. Run the following commands:

```
$ git checkout master
$ git branch --merged
```

Then, remove the outdated branches with:

```
$ git branch -d <name of local branch>
```

Note that the “-d” option is a safe delete. To delete abandoned branches you can use the following commands:

```
$ git branch --no-merged
$ git branch -D <name of branch>
```

Now, let’s remove the stale references to our remote branch. First, list the remote branches then prune them with the following commands:

```
$ git branch -r
$ git fetch --prune
```

**** IMPORTANT. Usually, you would NOT delete your branches after merging. When you delete branches you lose part of your commit history and therefore part of your project’s version history. ****

What To Turn In

You will submit the two images (a screenshot of your GitHub repo and a screenshot of the git log output on your terminal) from Part 3, Step 4 on Canvas.