Step-by-step Assignment Instructions for GitHub

Claire Duquennois

10/214/2020

Contents

1 What are Git and GitHub? 1

2 Getting started: 3
2.1 Starting an Account on GitHub . . . 3
2.2 Getting set up with Git . . . 3

3 Basic actions on GitHub 3
3.1 Creating a project repository . . . 3
3.2 Adding files to your repo via GitHub . . . 4
3.3 Working directly on GitHub in your repo . . . 4
3.4 Proposing edits in someone else's repo . . . 4
3.5 Accepting edits to your repo from someone else . . . 6
3.6 Making sure your forked repo is up to date with the parent repo . . . 6

4 Basic actions with Git 7
4.1 Cloning GitHub files to your computer . . . 7
4.2 Pulling GitHub changes to your computer . . . 8
4.3 Pushing local changes to your GitHub repo . . . 8
4.4 Keeping your local folder in sync with the parent repo through git . . . 9

5 Group work in a Public Repository (Problem Set 1) 10
5.1 Group leader: Forking the First Problem Set . . . 10
5.2 Group Team member: Forking the First Problem Set . . . 10
5.3 Group workflow on GitHub: . . . 10

6 Group work in a Private Repository (Problem Sets 2+) 11
6.1 Group "leader": Setting up the Problem Set Private Repo . . . 11
6.2 Group "leader": Adding the Problem set file. . . . 11
6.3 Group "leader": Adding collaborators . . . 11
6.4 Group team members: Becoming a collaborator on a private repo . . . 11
6.5 Group team members: Fork the private repo . . . 12
6.6 Group workflow on GitHub: . . . 12

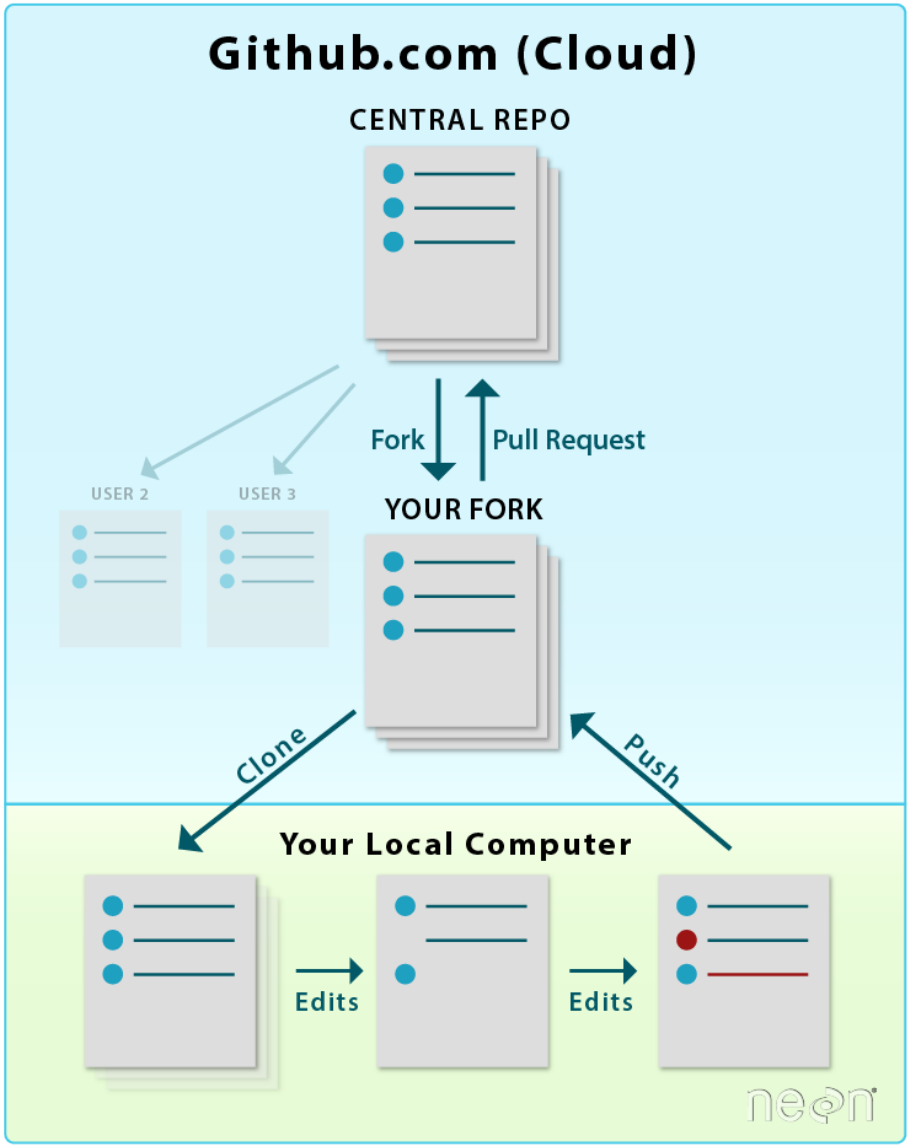
1 What are Git and GitHub?

Git and GitHub are often used in conjunction but are really two distinct things. Git is basically a version control software that is based on your local computer. GitHub is a webservice where you have an account that allows you to do version control via the website, which becomes particularly interesting when you are working on collaborative projects and want to avoid the issue of how to synchronize and manage changes to a

document when multiple people are working on the same document simultaneously. In theory you could use only one or the other but many people use them in conjunction.

To get a big picture overview of exactly what you can do with Git and Git hub, I would recommend taking the time to watch the Git and GitHub for Poets series of youtube videos https://www.youtube.com/playlist?list=PLRqwX-V7Uu6ZF9C0YMKuns9sLDzK6zoiV.

The diagram below give a basic sketch of what group workflow on GitHub should look like. Each group will have a central repo which each group member will fork onto their personal GitHub account. If you wish to work on your local computer, you can then use Git to clone, pull and push changes between your fork and your local machine. From your GitHub account, you can then integrate your code and sync your work with the central repo for your project.



2 Getting started:

2.1 Starting an Account on GitHub

This is quite straightforward:

1. Go to https://github.com/.
2. Click Sign up.
3. Choose a username, an email that GitHub will use to contact you and set you password
4. Respond to the invitation email to activate your account.

2.2 Getting set up with Git

Git is often used in conjunction with GitHub as a way to work on GitHub projects locally offline and then reintegrate your work back into a GitHub project. While there is a visual interface you can use to run Git, most programmers seem to prefer to run it directly by typing in Git commands through a terminal prompt.

On MACs: You can type Git commands directly into your Terminal (you can find it in Application/Utilities).

On Windows: You will want to install Git Bash. (See the following link for a video of the installation and some examples of commands https://www.youtube.com/watch?v=J_Clau1bYco.)

1. Go to git-scm.com and download the appropriate .exe file for your operating system.
2. Run the .exe file and install Git. In the installation, the only default you may want to change is to select the option to "Use Git and optional Unix tools from the Windows Command Prompt".

3. Once installed, launch Git Bash and you will see a terminal window in which you can execute Git commands.

You can use the terminal window to navigate through the folders in your computer's directory and then, using the proper commands, you will be able to use this terminal to push and pull materials from your computer onto github and vice versa. This will allow you to work on a github project on your local machine without having to be constantly online.

3 Basic actions on GitHub

3.1 Creating a project repository

Your GitHub account is basically composed of your project repositories (often referred to as repos). This is where your different projects will live. There are two main ways of creating a project repository: from scratch, or fork an existing repo.

3.1.1 From scratch:

To create a repo from scratch:

1. Log onto you GitHub account
2. Click on the **Repositories** tab
3. Click on the green **New** button which will create a new repository from scratch
4. Name your repository
5. Select "Public" (visible by anyone on the internet) or "Private" (only visible to you and any collaborators)
6. Select "Add a README" file (this is recommended. It is generally easier to work with a repo that is not empty).
7. Click the green **Create repository** button

3.1.2 By Fork

If you would like to copy someone's repo onto your own page so that you can freely work on it you can fork (ie copy) their entire repo

1. Navigate into a repository that you have access to (a public repository or one in which you are a collaborator)
2. Click the **Fork** button in the upper right corner

You should now have a copy of the repo on your account.

If you would like to rename the re-name the repo:

1. Open the repo
2. Go to the settings tab
3. Type in a new name and click Rename.

PRACTICE: Fork the course repo from claireduq/MQE_Causal_Inf to your account and rename it

3.2 Adding files to your repo via GitHub

1. Navigate into your repo
2. Click the **Add file** button and select Upload file in the drop down menu
3. Drop the file you wish to add into the upload space
4. Write a commit message like "adding the ... file"
5. Click the green **Commit changes** button

You should now see the file appear in the repository and, for certain types of files, if you select the file you can see the text or code you will be working with.

3.3 Working directly on GitHub in your repo

You can make edits to certain types of files that are in your GitHub repo directly from the GitHub website.

1. Navigate and open the file you with to edit
2. Click on the little pencil icon on the upper right of the file window
3. Edit the file
4. Add a commit description
5. Click the green **Commit changes** button at the very bottom of the page

Your file should now reflect the changes you made and if you look at the files history you should see the commit you just made.

3.4 Proposing edits in someone else's repo

You may find that you would like to propose edits to a file that lives in someone else's repo. There are two main ways to propose an edit to someone else's repo:

3.4.1 Editing directly in their repo

1. Navigate to the file in their repo
2. Click the small pencil icon in the upper right of the file window. This will automatically create a fork of the repo you are editing into your account.
3. Write your edits to the file
4. Scroll to the bottom of the page and add a commit message explaining your edits
5. Click **Propose changes**

6. Comparing change: you will now be shown a page that shows you how your edits fit into the existing document. Check to see the status of your edits:

   a. If it says "Able to merge":

   - Click the green button **Create pull request**
   - In the pull request window, add a description/name for your edits in your pull request
   - Click **Create pull request** at the bottom of the page You have now sent a pull request to the owner of the repo you edited. They will review you edits and decide if they wan to integrate them into their file.

   b. If it says "Can't automatically merge":

   - This means that your edits conflict with some other elements in the file. You can still create a pull request but the repo owner may need to make some modifications before integrating your changes.
   - Click the green button **Create pull request**
   - add a description/name for your edits
   - Click the green **Create pull request** button at the bottom of the page You have now sent a pull request to the owner of the repo you edited. They will need to review your edits, probably have to make some changes since there is a merge conflict, and decide if they wan to integrate them into their file.

PRACTICE: Add your first name and username to claireduq/MQE_Causal_Inf/GitHub_names.txt.

3.4.2 Editing a file in your repo that was forked from someone else's repo and initiating a pull request

If you forked a repo from someone else, made some changes that you liked, and would like to propose those changes to the owner of the original repo you can also do this via a pull request. Note: This will generate a pull request for ALL of the commits (changes) you made to any file in your forked repo since you originally forked it.

1. Navigate to your forked repo
2. Click the **Pull requests** tab.
3. Click the green **New pull request** button
4. Comparing change: you will now be shown a page that shows you how your edits fit into the existing repo. Check to see the status of your edits:

   a. If it says "Able to merge":

   - Click the green button **Create pull request**
   - In the pull request window, add a description/name for your edits in your pull request
   - Click **Create pull request** at the bottom of the page You have now sent a pull request to the owner of the repo you edited. They will review you edits and decide if they wan to integrate them into their file.

   b. If it says "Can't automatically merge":

   - This means that your edits conflict with some other changes in the file. You can still create a pull request but the repo owner may need to make some modifications before integrating your changes.
   - Click the green button **Create pull request**
   - add a description/name for your edits
   - Click the green **Create pull request** button at the bottom of the page You have now sent a pull request to the owner of the repo you edited. They will need to review your edits, probably have to make some changes since there is a merge conflict, and decide if they wan to integrate them into their file.

3.5 Accepting edits to your repo from someone else

If someone wants to propose changes to a file in one of your repos they will generate a pull request. Unless they are a collaborator on the repo you will need to review the changes and approve them before they are pulled into your file.

You will know that a pull request was made as you will

1. Maybe receive an email informing you
2. When you navigate to the repo in question the pull request will show up as a number next to the **pull request** tab.

Reviewing a pull request:

1. Click on the **pull request** tab in your repo
2. Click on the pull request you would like to review (it will be named according to the description given by it's originator)

There are two possibilities when you receive a pull request. The edits proposed may be easy to automatically merge into your file or not.

a. If the pull request is able to Merge:
   - If you want to see the detail of what edits were made, click on pull request name to look over the changes and then navigate back to the main pull request page
   - Click the green **Merge pull request** button
   - Click the green **confirm merge**

b. If there are conflict in the pull request:
   - There will be a warning sign informing you of a conflict.
   - Click the **resolve conflicts** button
   - GitHub will open an editable window that contains your file in which the conflicting part(s) of the file(s) is highlighted by a red bracket. Within these brackets the proposed changes are bracketed by the lines [«««< master] and [=======] and what is currently in your file is bracket by [=======] and [»»»> master]. With all this information, you can edit your file within this window until you are happy with it.
   - Click the **Mark as resolved** button (upper right of the file window)
   - Click the green **commit merge** button
   - Click the **I understand, update master** button in the pop up window
   - Click the green **Merge pull request** button
   - Click the green **Confirm merge** button

The pull request will now switch to Merged status. If you navigate to the file in your repo, you should see the new edits appear. And of you go to the file's **history** (button on the upper right) you can see the commit that was just made via the merged pull request.

3.6 Making sure your forked repo is up to date with the parent repo

If you are collaborating on a project, before you start working on the files you will want to make sure that you are working on the most up to date version of your group's code. Similarly, if you are referring to the course notes in your repo, you want to make sure that these notes are the most up do date version and reflect any changes or edits I made to them since you forked the course repo. Keeping your repos up to date will reduce the risk of merge conflicts and make your group collaboration much smoother.

3.6.1 How do I know if my repo is not up to date?

To check in see if someone made changes to the parent repo you originally forked your repo from that are not reflected in your fork:

- Navigate into your repo

- You will see a cell above your files with a statement like:

  `This branch is 16 commits behind claireduq:master`

  this means your repo it not up to date.

3.6.2 How to sync your repo if it is not up to date:

1. Navigate to the main page of your forked repository
2. Click on the **pull request** tab
3. Click the green **New pull request** button
4. You will see a cell that specifies the base repository and the head repository. We need to have the base repository be your repository and the head repository be the parent repository. For example, if you are trying to update your fork of the course repo we want

   `base repository: student/MQE_Causal_Inf` and `head repository: claireduq/MQE_Causal_Inf`

   to do this you can likely simply click on the highlighted blue text "switching the base" that is in the main window under the cell that specifies the base and the head.
5. Click the green **Create pull request** button
6. Name your pull request
7. Scroll down and click the green **Create pull request** button
8. Click the green **Merge pull request** button
9. Click the green **Confirm Merge** button
10. Check and see that your files have updated and are now in sync with the parent repo

Note: this can be set up tobe done with a lot less clicking using git (see below).

PRACTICE: Update your forked MQE_Causal_Inf repos so that the reflect my recent edits.

4 Basic actions with Git

While you can make edits directly on GitHub, if you are making substantial changes, or want to work offline, you will want to work on your local machine. This is especially true if you are writing code. You cannot run your code to make sure that it runs correctly in GitHub. So when you are making substantial changes to an R code file you will want to "Pull" the file off of GitHub, work on it in R on your local machine, make sure it runs correctly, and then push the changes back to GitHub. This can be done fairly easily using Git.

4.1 Cloning GitHub files to your computer

To copy the files in a repo to your local machine:

1. Make an empty folder named, for example, PS1_GS where you will put the files you are going to pull off GitHub (preferably in Dropbox or someplace where it will be backed up).
2. Open your command terminal (or Git Bash on Windows).
3. Navigate via your command terminal to the new folder by typing

   `$ cd /c/Users/ *******/Dropbox/PS1_GS`

   followed by enter.

   Note: the directory path will be different for you! To easily get your directory path, you can simply drag and drop the folder into your terminal window.

4. Check that you are in the right folder by typing

   ```
   $ pwd
   ```

   and you should see the directory path to the folder you want to be using.

5. Suppose your GitHub user id is `myuserid` and your repo you would like to clone is `PS1_name1_name2_name3`. In your github repo, click the green code button and copy the link it gives you. Most likely it will look like `https://github.com/myuserid/PS1_name1_name2_name3.git` (Notice, the url is just the web address with .git added to the end.)

6. In the terminal type `git clone` and copy the url it gave you

   ```
   $ git clone https://github.com/myuserid/PS1_name1_name2_name3.git
   ```

   followed by enter.

You should now find that the folder you created has a copy of the group's R-Markdown file.

4.2 Pulling GitHub changes to your computer

If a change is made to your GitHub repository and you would like the changes to be reflected in your cloned local folder, in the terminal type

```
$ git pull
```

4.3 Pushing local changes to your GitHub repo

Once you have made changes to the repo file on your computer, and saved them, you may want to have your GitHub repo reflect these changes.

1. Go to the terminal window and navigate to your cloned directory by typing

   ```
   $ cd /c/Users/ *******/filepath
   ```

   ---

2. **WARNING:** At this point you want to check if GitHub knows who you are. Type

   ```
   $ git config --list
   ```

   and it will spit out a bunch on information. Look and see if your GitHub username and the email you used to sign on to GitHub with are listed. If they are all is well. If they are not listed type

   ```
   $ git config --global user.name "janedoe"
   ```

   ```
   $ git config --global user.email janedoe@pitt.edu
   ```

   Make sure you set these to your github user name and the email you used to sign up with GitHub.

   ---

3. Check the status of your edits. Type

   ```
   $git status
   ```

   You should see:

   - if you have modified files with edits that have not yet been commited.
   - if you have any new files that do not exist in the GitHub repo

4. If you have new files type

   ```
   $ git add .
   ```

   to add all new files to GitHub

5. To prepare your modifications to be committed, type

   ```
   $ git commit -a -m "Jane Doe's changes"
   ```

   where the -a argument tells git to commit all the modifications and -m adds a message to you commit (a good practice when working on jointly edited files). Now if you type

   ```
   $ git status
   ```

   you will see that the files you changed are ready to be pushed to github so that they are reflected there.

6. Now you can type

   `$ git push origin master` or just `$git push`

   which basically tells git to push your changes from your remote "origin" computer to the "master" branch on github.

7. You can now look at your repository on github and check that the edits you made appear in your files and the files history.

4.4 Keeping your local folder in sync with the parent repo through git

To make sure a folder on your computer reflect the most recent changes made in an upstream repo (that you originally forked from) you need to configure the folder so that you can easily `fetch` any changes in the upstream repo:

1. Go to the terminal window and navigate to the git folder by typing

   ```
   $ cd /c/Users/ *******/filepath
   ```

2. List the current configures remote repository for your folder: `$ git remote -v`

   ```
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   ```

   ```
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
   ```

3. We want to change this. Specify a new remote upstream repo that will be synced with the fork

   ```
   $ git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
   ```

4. Verify the new upstream repository you've specified for your fork.

   ```
   $ git remote -v
   ```

   ```
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   ```

   ```
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
   ```

   ```
   > upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
   ```

   ```
   > upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
   ```

5. You can now `fetch` changes to that repo and have them reflected on your local machine with the command

   ```
   $ git fetch upstream
   ```

6. Check out your fork's local master branch.

   ```
   $ git checkout master
   ```

   Note: you may need to replace the term `master` with `main` if you are a collaborator on the upstream repo.

7. Merge the changes from upstream/master into your local master branch. This brings your fork's master branch into sync with the upstream repository, without losing your local changes.

   `$ git merge upstream/master`

   Notes:

   - Git may send you to a weird window asking you to write a commit message. You can escape this with `Shift+:` followed by Q.
   - you may need to replace the term `master` with `main` if you are a collaborator on the upstream repo.

5 Group work in a Public Repository (Problem Set 1)

For the first problem set you will be working in a public repository. Keep in mind that this means that your repository will be visible on the internet. I recommend that you select a group leader who's repo will be the central repo for the group where you combine your joint work.

5.1 Group leader: Forking the First Problem Set

1. Navigate to claireduq/MQE_PS1
2. Fork the problem set repo (see instructions above)
3. Rename the problem set repo: PS1_name1_name2_name3 (see instructions above)

5.2 Group Team member: Forking the First Problem Set

1. Navigate to groupleader/PS1_name1_name2_name3
2. Fork the problem set repo (see instructions above)

5.3 Group workflow on GitHub:

Now that you all have repos with the problem set, you will want to:

1. Use Git to clone your repos to your local machines (see instructions above).
2. Work on your contribution to the code and answers on your local machine
3. Make sure your edits run by knitting the .Rmd file
4. Remove all files other than the .Rmd file from your local repo folder
5. Use git to push your changes to your GitHub repo (see instructions above).
6. **If you are a team member:**

- Submit a pull request to your group leader so she can incorporate your contribution into the main project repository (see instructions above).

  If you are a team leader:

- Review your team member's pull requests and merge them into your main project repo

7. Next time you want to work on the assignment, make sure you syncronize your forked repo with the group leaders repo so that you are working on the most up to date version of your groups code (see instructions above). `$git pull` any updates to your local machine and repeat steps 2-6.
8. Once the main group .Rmd file is finalized, you can compile it to html, pdf or word for submission.

Hints: I would recommend only keeping track of the .Rmd file on GitHub, until the very end when you will produce your final html, pdf or word file. This will make it so that when you deal with any conflicts in pull requests, you only need to edit the .Rmd file, and don't have to worry about the other files.

When you are working on your local machine, do make sure to knit the .Rmd file frequently to make sure your code is compiling correctly and that you can produce the html or pdf files. However when it comes time to push your changes to GitHub, I would recommend deleting all the extra files that R created so that you only push the .Rmd file and only have to worry about merging the .Rmd file when you make pull requests to your group leader.

6 Group work in a Private Repository (Problem Sets 2+)

The group leader is not going to play such a central role when working on a group project in a private repo. This is because all of the team members will be collaborators in this repo and will have similar editing privileges. **This means that any changes any group member makes to the project can be merged into the main file by any team member (not necessarily the group leader who initiated the repo).** Thus the group leaders unique responsibilities are mostly limited to setting up and hosting the private repo you will work in.

6.1 Group "leader": Setting up the Problem Set Private Repo

To set up a private repo:

1. Log onto you GitHub account
2. Click on the **Repositories** tab
3. Click on the green **New** button which will create a new repository from scratch
4. Name your repository "PS2_name1_name2_name3"
5. Select "Private"
6. Select "Add a README" file
7. Click the green **Create repository** button

Congratulations! You have create your repository! A couple things to note: Private repos will not show up on your main home site. To navigate to them you need to go to the Repository tab.

6.2 Group "leader": Adding the Problem set file.

1. Download the Problem Set .Rmd file from Canvas
2. Navigate into the private PS2_name1_name2_name3 repo
3. Click the **Add** file button and select Upload file in the drop down menu
4. Drop the .Rmd file you downloaded into the upload space
5. Write a commit message like "adding problem set Rmd file"
6. Click the green **Commit changes** button

You should now see the .Rmd file appear in the repository and if you select the file you can see the .Rmd code you will be working with.

6.3 Group "leader": Adding collaborators

1. Navigate into the private PS2_name1_name2_name3 repo
2. Click on the **Settings** tab
3. Click on the **Manage access** tab in the left menu
4. Click on the green **Invite collaborator** button
5. Enter your teammates username in the popup box, select the correct user and invite them
6. Repeat for all of your teammates, Erica (els204) and I (claireduq)

6.4 Group team members: Becoming a collaborator on a private repo

1. Once invited, you will receive an invitation via email to collaborate on the repo (it could be in the updates tab in gmail).

2. Click view the invitation to be sent to the repo
3. Once in the repo, click the green **Accept invitation**
4. You can now view the private repo your leader set up.

6.5 Group team members: Fork the private repo

If you want to be able to use Git in order to clone the repo to your computer's files you will need to generate a fork of the repo (see instructions above).

6.6 Group workflow on GitHub:

Now that you all have repos with the problem set, you will want to:

1. Use Git to clone your repos to your local machines (see instructions above).
2. Work on your contribution to the code and answers on your local machine
3. Make sure your edits run by knitting the .Rmd file
4. Remove all files other than the .Rmd file from your local repo folder
5. Use git to push your changes to your GitHub repo (see instructions above).
6. Here is where things will be different as a collaborator in a private repo (as opposed to contributing to a public repo):

If you are a team member:

Submit a pull request to incorporate your contribution into the main project repository **AND** Accept/Merge the edits you proposed your repo into the main project repository (**Notice the difference here:** since you are collaborators you can authorize and merge pull requests into the main folder, you do not need the repo owner (group leader) to do this.) You can do this by

a. Navigate to your forked repository that contains the changes you want to integrate into the main repo
b. Click on the **pull request** tab
c. Click the green **New pull request** button
d. Click on the green **Create pull request** button
e. Name your pull request
f. Click on the green **Create pull request** button
g. From here there are two possibilities:

- If the pull request is able to Merge:
  - If you want to see the detail of what edits were made, click on pull request name to look over the changes and then navigate back to the main pull request page
  - Click the green **Merge pull request** button
  - Click the green **confirm merge**
  - Your changes have now been added to the main file
- If there are conflict in the pull request:
  - There will be a warning sign informing you of a conflict.
  - Click the **resolve conflicts** button
  - GitHub will open an editable window that contains your file in which the conflicting part(s) of the file is highlighted by a red bracket. Within these brackets the proposed changes are bracketed by the lines [«««< master] and [=======] and what is currently in your file is bracket by [=======] and [»»»> master]. With all this information, you can edit your file within this window until you are happy with it.
  - Click the **Mark as resolved** button (upper right of the file window)
  - Click the green **commit merge** button
  - Click the **I understand, update master** button in the pop up window
  - Click the green **Merge pull request** button
  - Click the green **Confirm merge** button
  - Your changes have now been added to the main file

If you are the group leader:

- PRO: You no longer need to review your group's pull requests!
- CON: Things could be changing in your repo without you realizing it! Keep an eye on your repo history so you know what happened while you were away. You may also want to be cautious when pulling repo changes to your local machine since there may be changes you didn't know about. (Maybe keep a back up of your own work on your local computer for your records?)

7. Next time you want to work on the assignment, make sure you syncronize your forked repo with the group leaders repo so that you are working on the most up to date version of your group's code (see instructions above). `$git pull` any updates to your local machine and repeat steps 2-6. .
8. Once the main group .Rmd file is finalized, you can compile it to html, pdf or word for submission.

Hints: I would recommend only keeping track of the .Rmd file on GitHub, until the very end when you will produce your final html, pdf or word file. This will make it so that when you deal with any conflicts in pull requests, you only need to edit the .Rmd file, and don't have to worry about the other files.

When you are working on your local machine, do make sure to knit the .Rmd file frequently to make sure your code is compiling correctly and that you can produce the html or pdf files. However when it comes time to push your changes to GitHub, I would recommend deleting all the extra files that R created so that you only push the .Rmd file and only have to worry about merging the .Rmd file when you make pull requests to your group leader.