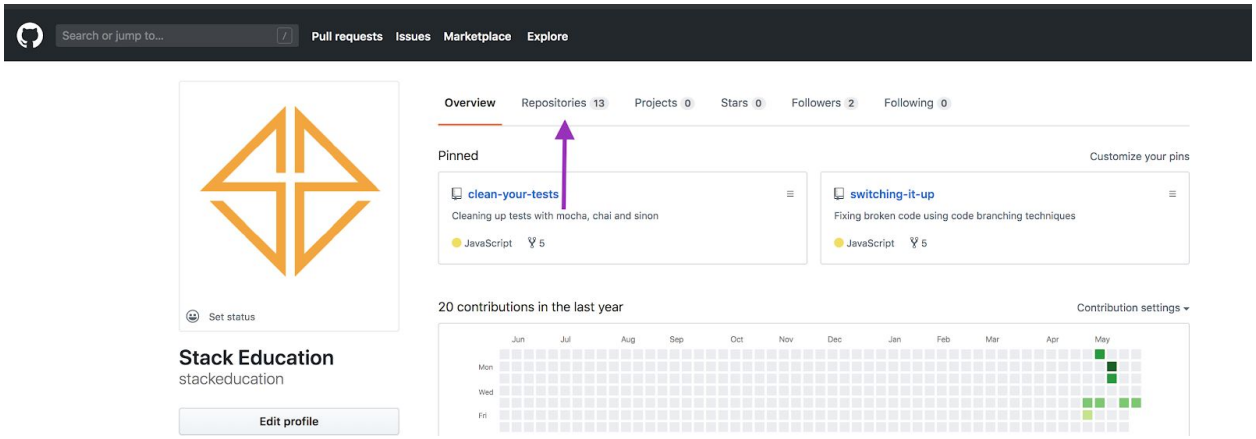
Exercise Setup Guide

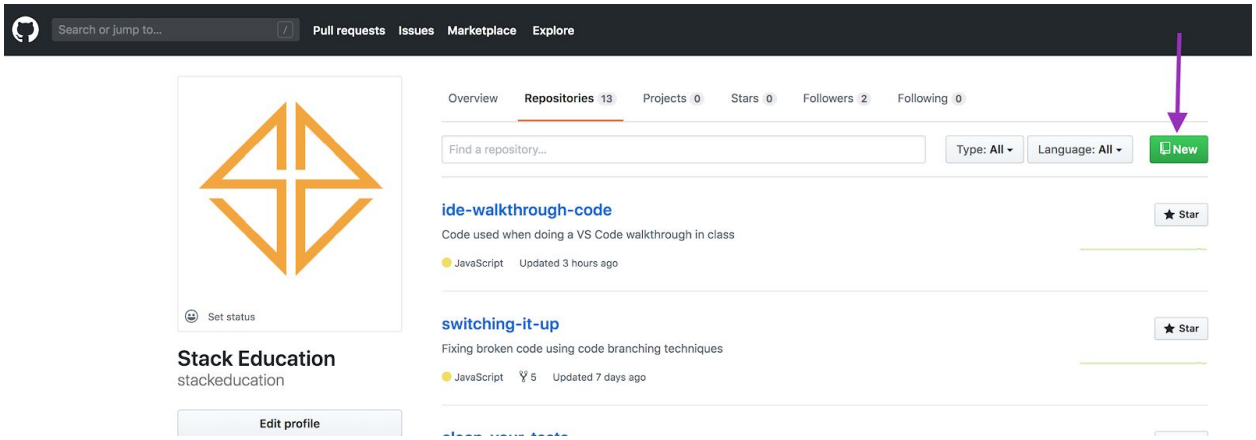
This guide is meant to walk you through the steps to get your Git repository setup for each exercise you work on. Git is a habit and so it is expected that all exercises are submitted as pull requests for review by the instructor. This guide should make this process easier as you get used to working with Git. Within a few weeks you will find you don't even need it anymore but this will make the time between now and then less stressful.

Create a New Repo in GitHub

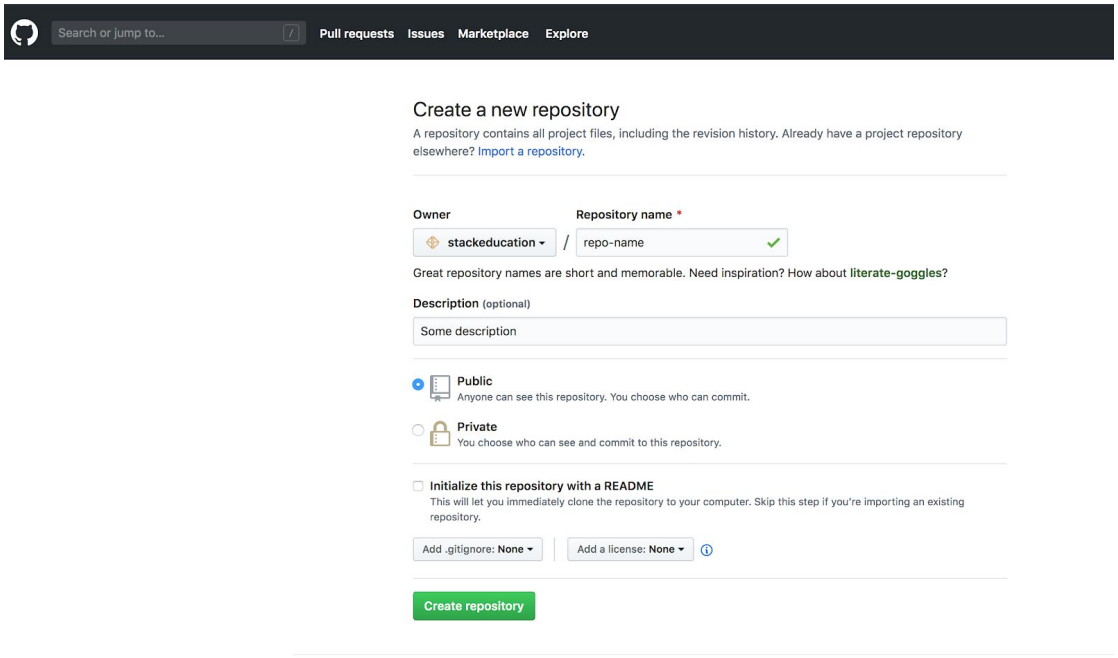
Start by visiting your GitHub profile, from here you will click on the Repositories link.



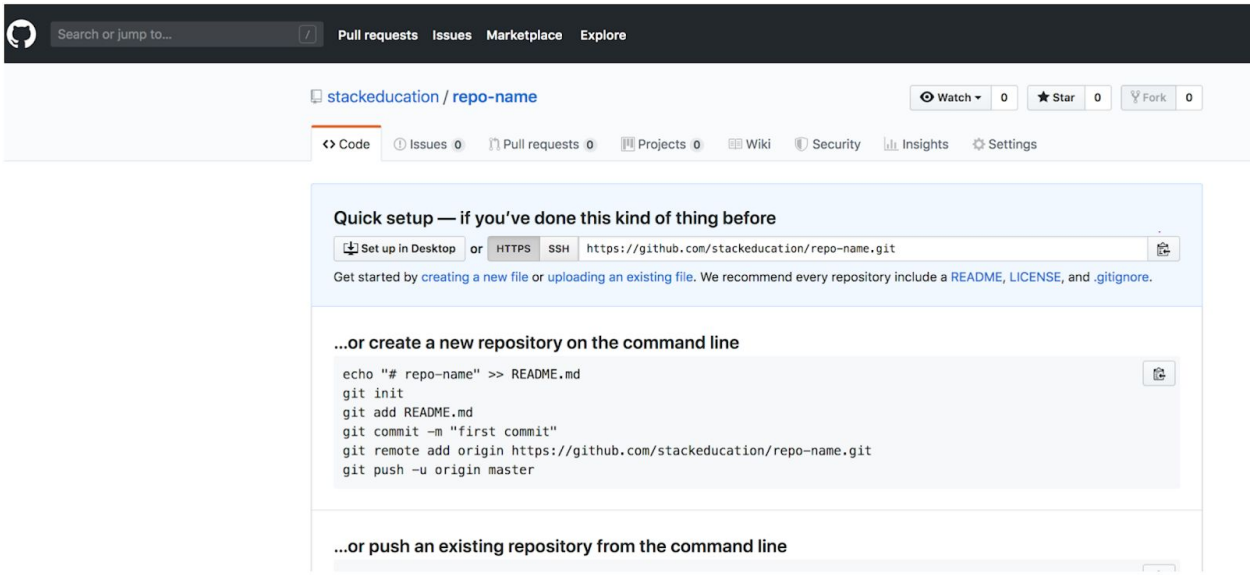
On the Repositories page you will use the green New button to start the process of creating a new repository.



On the repository creation page you will fill in the name of your new repository, use the name provided on the exercise slide for consistency sake. A description is optional. The rest of the settings on this page should be left as the defaults. You can now click the green Create repository button.

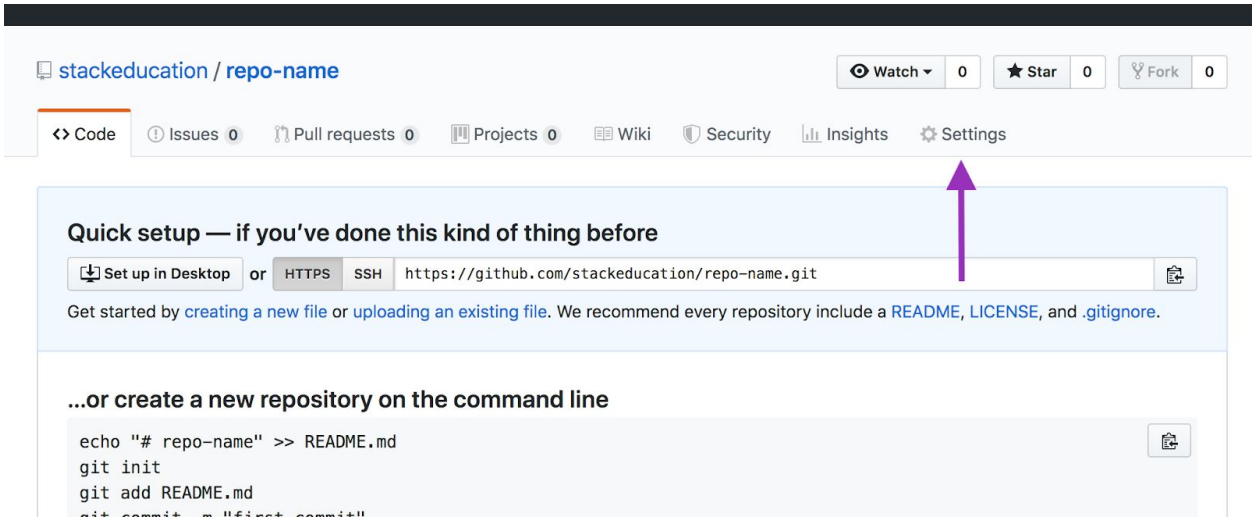


This will bring you to the page for your new repository. It is currently empty as expected.

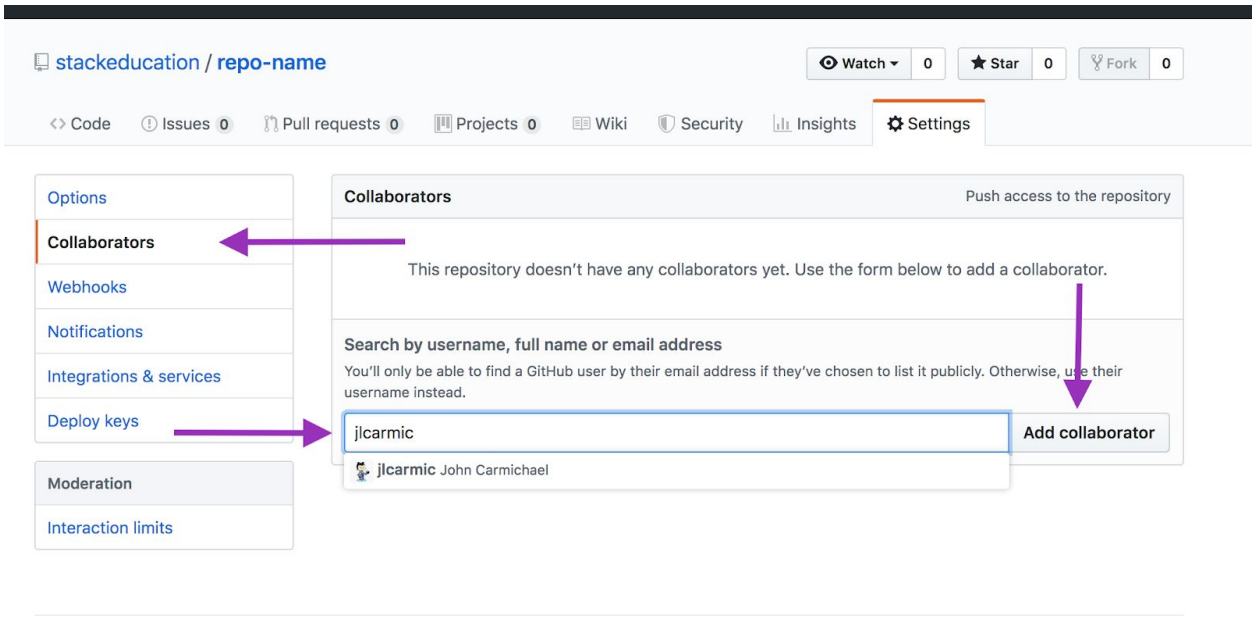


Make Your Instructor a Collaborator

In order for your instructor to add code to your repo they will need to be a collaborator for your new repository. You can add them as one by clicking on the Settings tab.

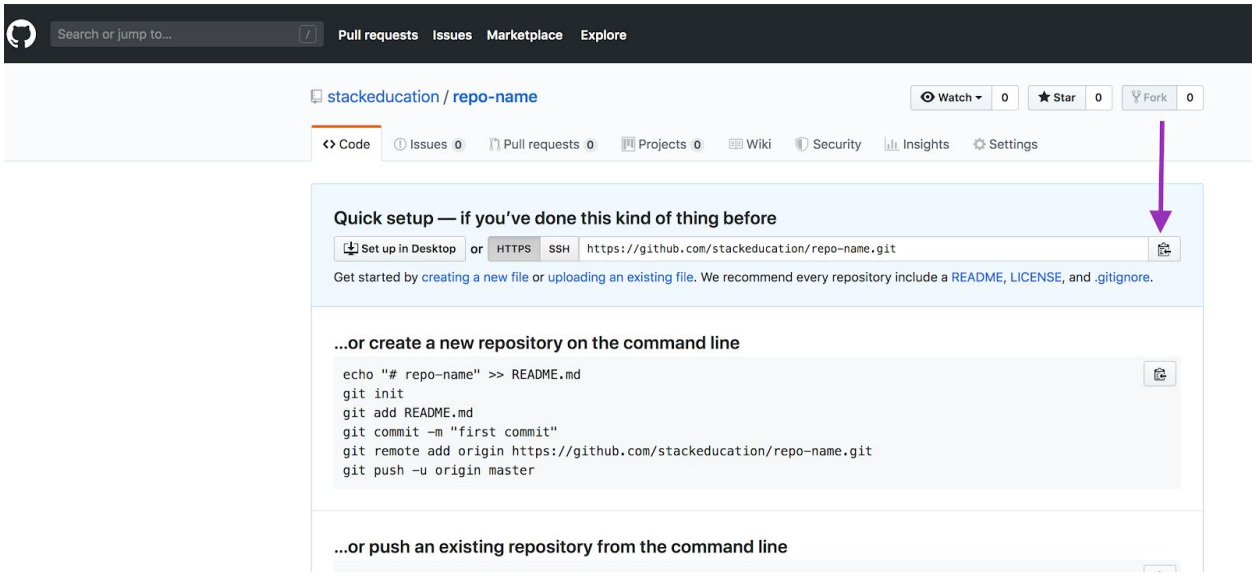


Next click on "Collaborators" in the side menu, type the instructors GitHub name into the text box and click the "Add collaborator" button.



Clone Your New Repository Locally

After creating our new repo and adding your instructor as a collaborator, we need to clone it to our machine so we can work on it locally. We use the address we've copied from the repository page to do this.



With this address we can now use the "git clone" command to create a copy of our repo locally. Cloning a repo will create a folder for it in the location we are in so be sure to navigate to the folder where you'd like to keep your code before issuing this command.

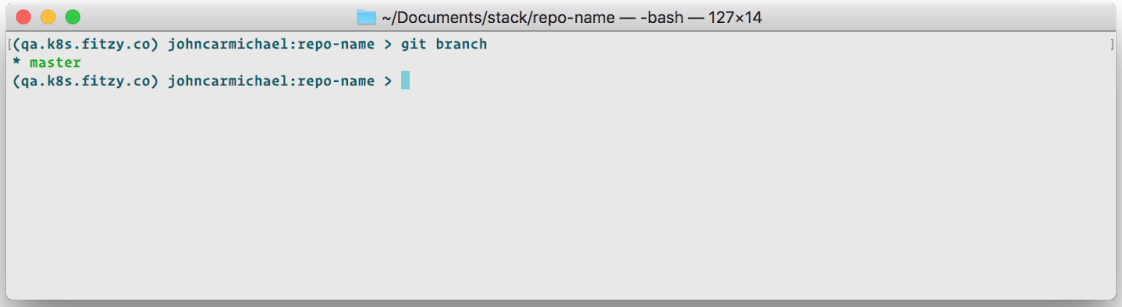
When we use "git clone" the git program fetches all of the files, branches and history from GitHub for us. It also sets up a remote called "origin" for us which will point back to the GitHub repository from which we cloned. This link is helpful as we need it later to push code back up to GitHub.

Once the clone is complete we can switch into that folder using the “cd” (change directory) command.

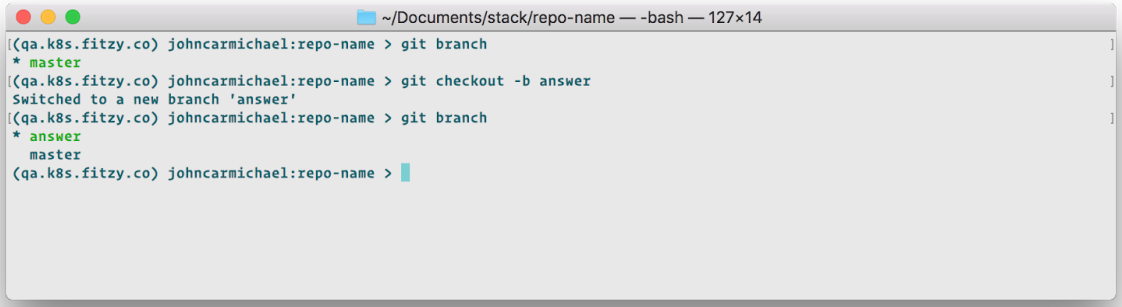
If you clone before the instructor has pushed code into your repository you will get a message about cloning an empty repository. This is ok, once the instructor has pushed the code you can issue a `git pull` command to fetch the latest code.

Create a Branch to Work In

In our newly cloned repo we can use the “git branch” command to see that we are currently on the master branch and it is the only branch we have.



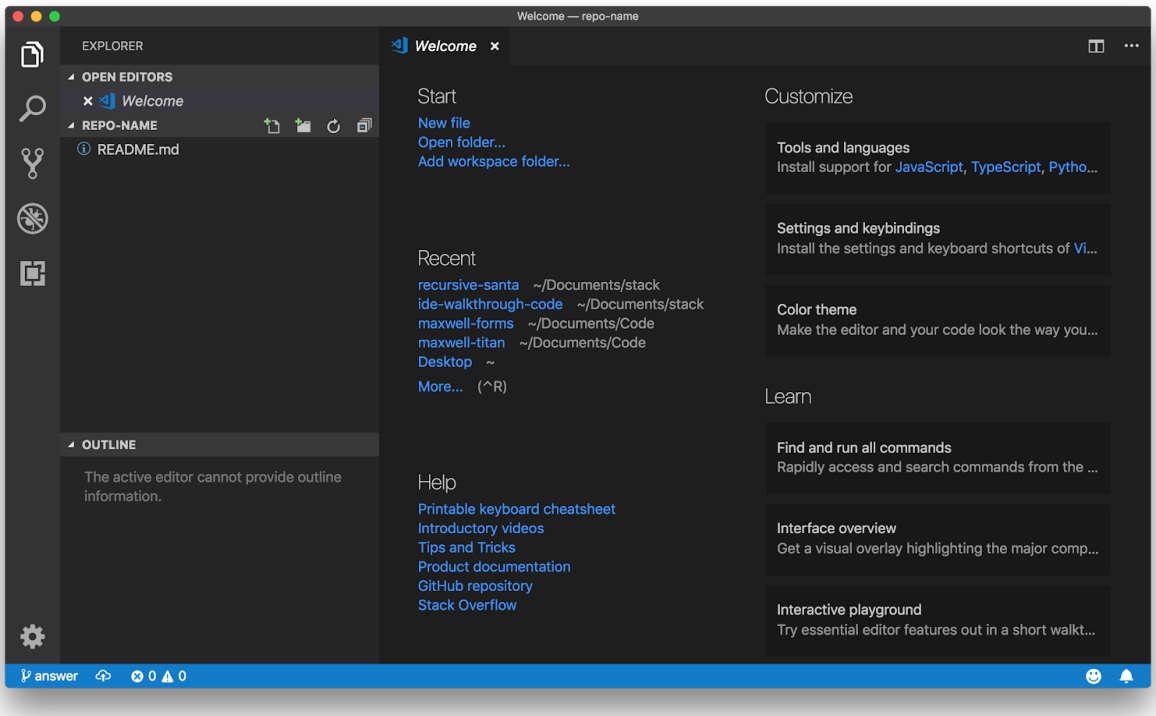
We will want to create a new branch to contain our work we for this exercise. We can do that with the “git checkout” command. This command allows us to move between branches and if we pass it the “-b” flag it will even create a new branch and switch us to it.



Now we can begin working on the assignment which will typically require working in VSCode. Fortunately VSCode ships with the “code” command for opening the program. If we call this command and pass it a “.” it will open VSCode within the current folder. This is because on the command line “.” means “current folder.”

```
~/Documents/stack/repo-name — -bash — 127×14
(qa.k8s.fitzy.co) johncarmichael:repo-name > git branch
* master
(qa.k8s.fitzy.co) johncarmichael:repo-name > git checkout -b answer
Switched to a new branch 'answer'
(qa.k8s.fitzy.co) johncarmichael:repo-name > git branch
* answer
  master
(qa.k8s.fitzy.co) johncarmichael:repo-name > code .
```

If you are on a Mac and the code command didn't work for you, go see the `VS Code Configuration Guide` for the steps you need to get it working on your machine.



Now go code all the things!

Add Your Changes to the Staging Area

Ok so you have some changes you want to hang on to, like a video game save point. You will likely do this many times while working on the assignment but definitely at least once at the end when submitting. We can see the files that have been changed either in VSCode or by running the “git status” command.

```
~/Documents/stack/repo-name — -bash — 127×14
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
(qa.k8s.fitzy.co) johncarmichael:repo-name >
```

Here we see I have modified files that I have not staged yet. To stage them I simply run the “git add” command with the file name. After that running the “git status” command will show me that they are indeed staged. Note, they are not committed yet, only staged to be committed.

```
~/Documents/stack/repo-name — -bash — 127×19
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
(qa.k8s.fitzy.co) johncarmichael:repo-name > git add index.js
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.js

(qa.k8s.fitzy.co) johncarmichael:repo-name >
```

Commit Your Changes

We now have staged files that we want to add to our repo locally. Again, doing this staging/committing process often as you work on code is a good habit. Remember, it's all local and in a branch so you can't break anything, master is still unchanged. To add these files to our local repo we use the "git commit" command. When we do commit we always want to add a message that tells us and others what exactly the changes are in the commit. We add a message with a commit by providing the "-m" flag.

```
~/Documents/stack/repo-name — -bash — 127×24
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
(qa.k8s.fitzy.co) johncarmichael:repo-name > git add index.js
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.js

(qa.k8s.fitzy.co) johncarmichael:repo-name > git commit -m "I fixed a typo"
[answer 2355c5d] I fixed a typo
 1 file changed, 1 insertion(+)
(qa.k8s.fitzy.co) johncarmichael:repo-name >
```

Our changes are now part of the local repo and we can continue working on more changes or we can move on to pushing those changes up to the remote on GitHub.

Push Your Updated Branch to GitHub

To get our branch and all of its changes up to GitHub we will use the “git push” command. This command needs two pieces of information.

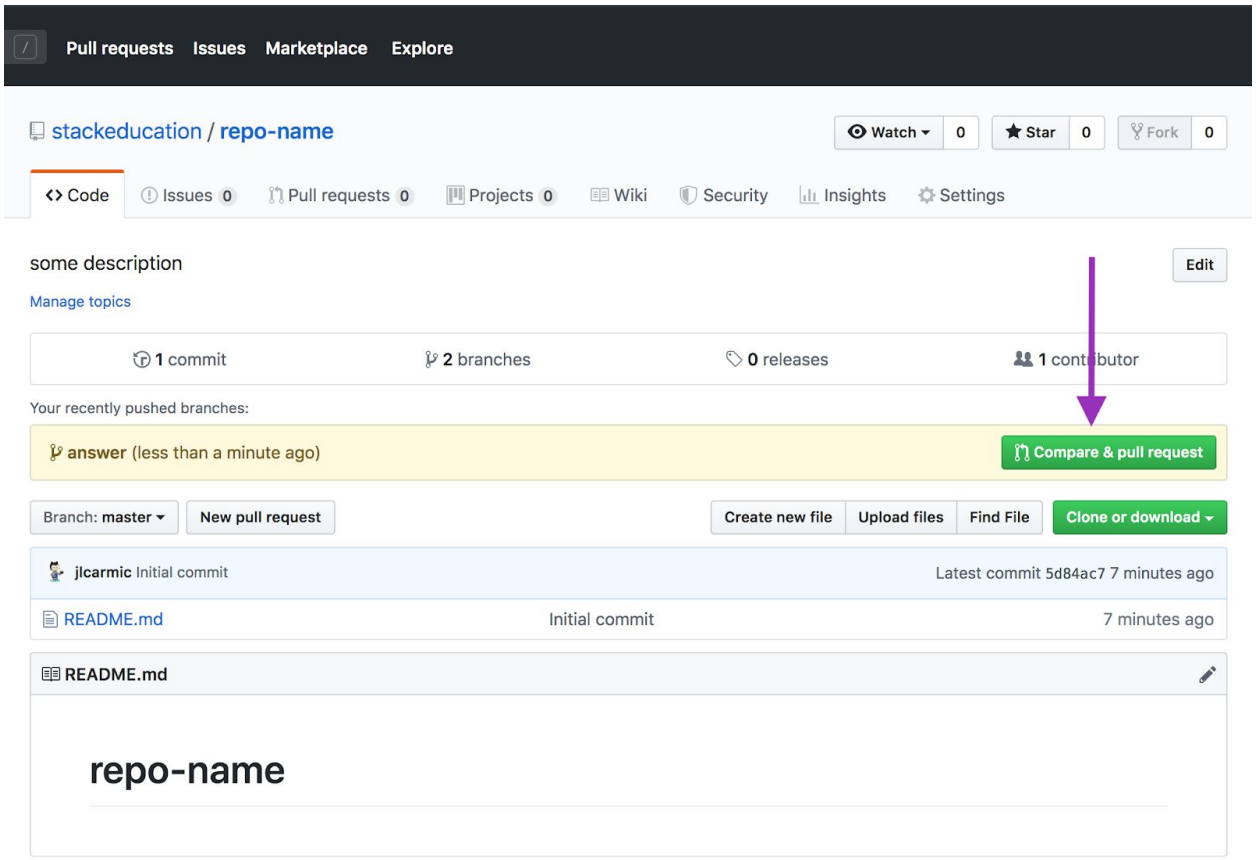
The first is the remote repository we want to push to. Remember when we cloned the repo git very nicely set up a remote for us called "origin" which pointed back to the repo we cloned. We can now use that shorthand “origin” to tell git that we want to push to that location.

The second piece of information is the branch we want to push. We can push any branch we have locally, in this case we are working in a branch we created called “answer” so that is the branch name we provide when pushing. Running this command will push the answer branch and all of its history from our local machine and store it in GitHub for us.

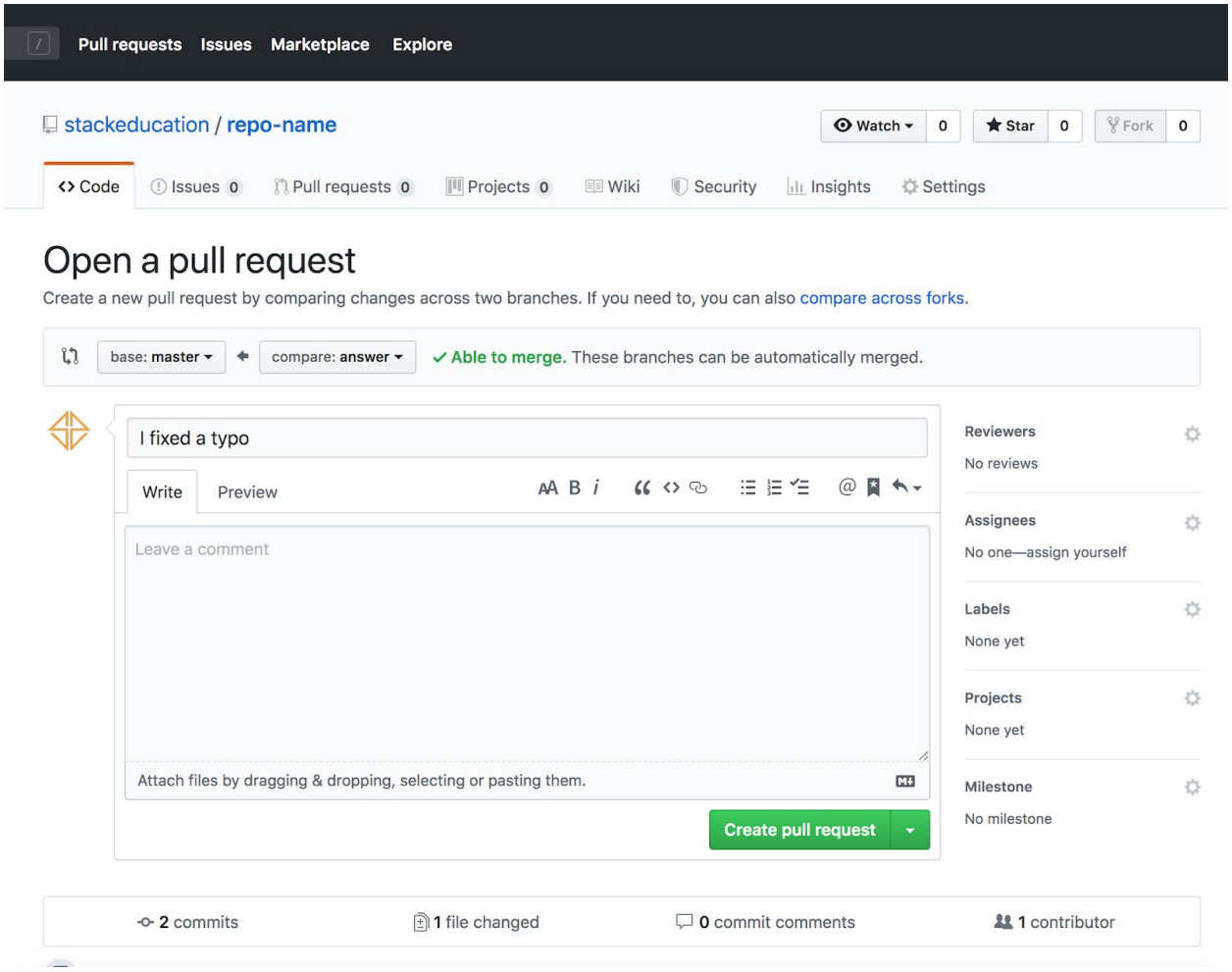
```
~/Documents/stack/repo-name — -bash — 127×24
(qa.k8s.fitzy.co) johncarmichael:repo-name > git status
On branch answer
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.js

(qa.k8s.fitzy.co) johncarmichael:repo-name > git commit -m "I fixed a typo"
[answer 2355c5d] I fixed a typo
 1 file changed, 1 insertion(+)
(qa.k8s.fitzy.co) johncarmichael:repo-name > git push origin answer
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 492 bytes | 123.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'answer' on GitHub by visiting:
remote:      https://github.com/stackeducation/repo-name/pull/new/answer
remote:
To https://github.com/stackeducation/repo-name.git
 * [new branch]      answer -> answer
(qa.k8s.fitzy.co) johncarmichael:repo-name >
```

Create a Pull Request in GitHub

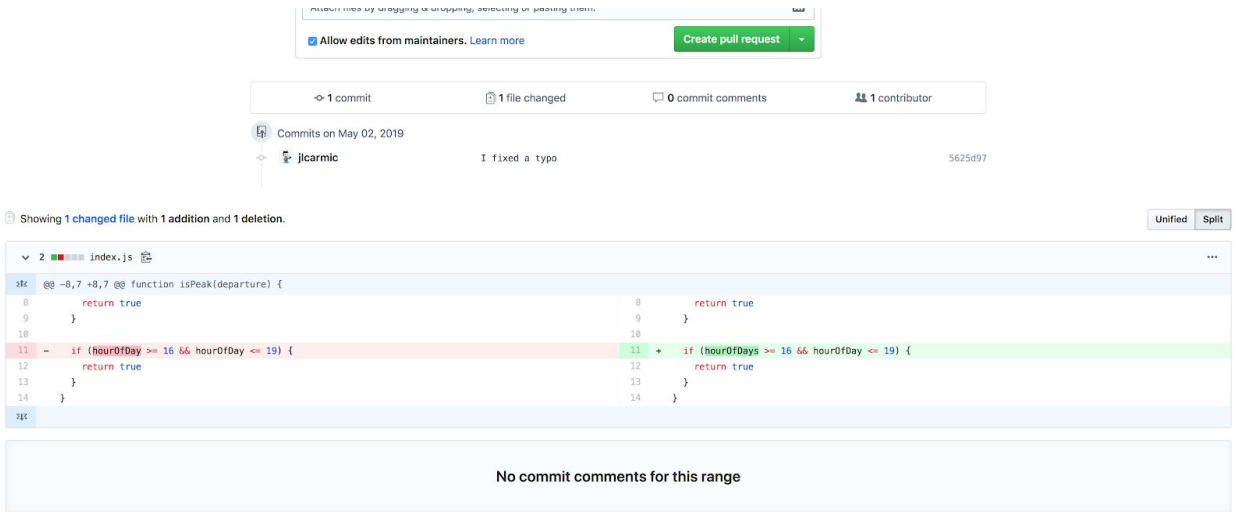
When we visit our repo on GitHub after pushing our branch we see that GitHub knows this has happened and provides us with a handy link to create a pull request.



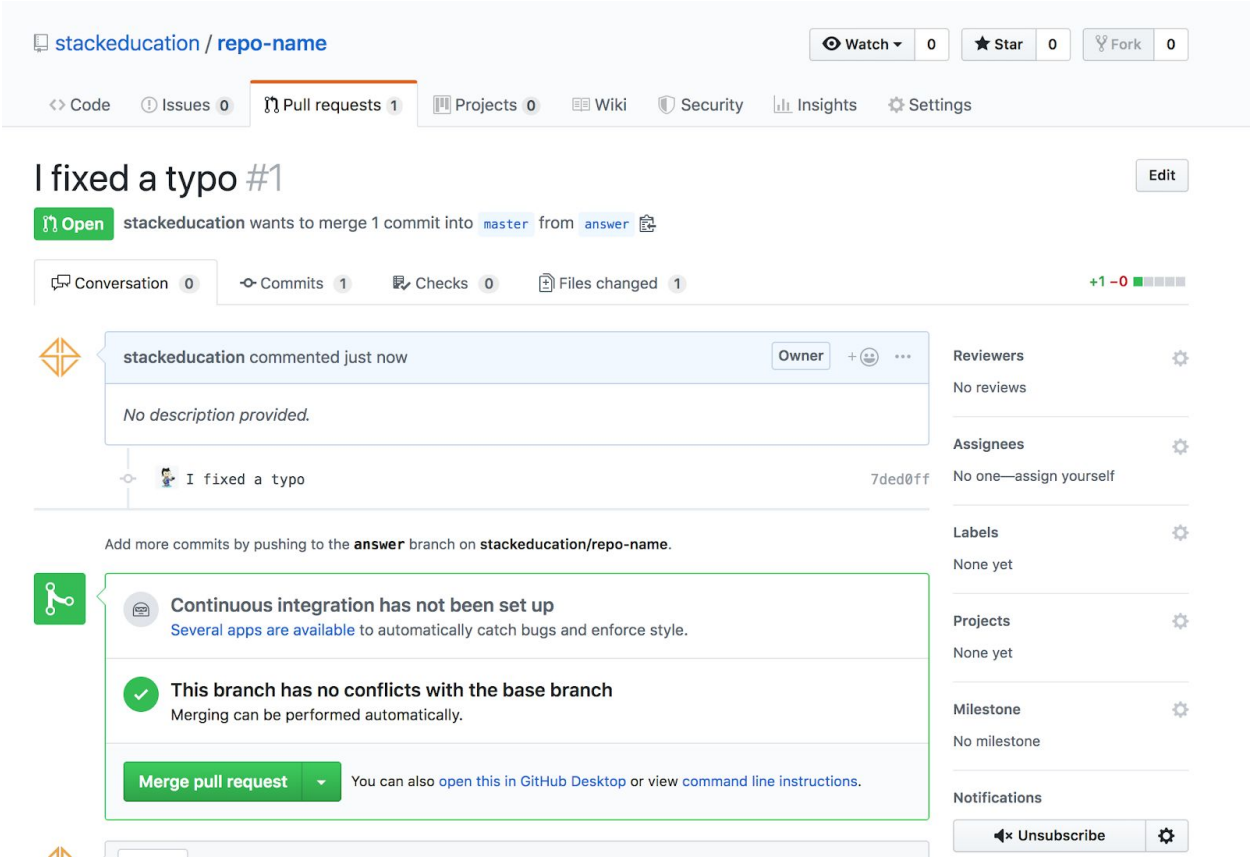
Clicking that link brings us to a page that lets us create a new pull request. Here we can provide a title and some context about our changes so that the people viewing them have more context. Notice that it auto-fills with our most recent commit message for us.



And if we scroll down we can see our changes laid out for us to double check before creating the request. The left side here will show the original code and the right will be our changes. Deletions will be highlighted in red and additions highlighted in green. We also see the count of files changed which can be helpful when our branch has a lot of changes. Also each commit will be listed.

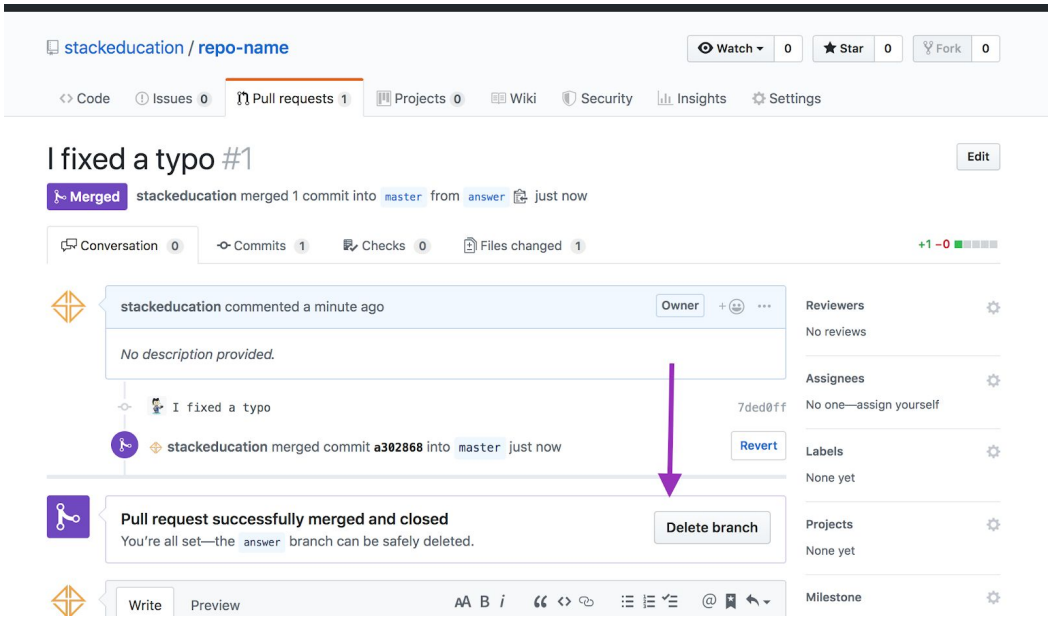


Clicking create will bring us to our newly created pull request. Here we can assign the request to ourselves and anyone else who may have helped. We can also request reviews from specific people. Other developers on our team can make comments on our code and provide feedback as well as request changes from this interface. **Make sure you request a review from the instructor.**

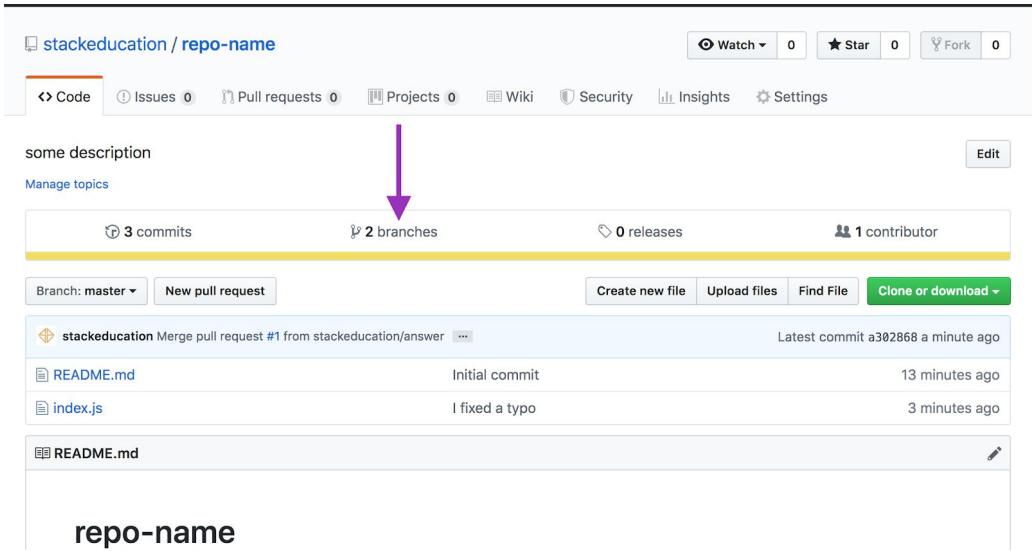


Deleting A Branch Remotely and Locally

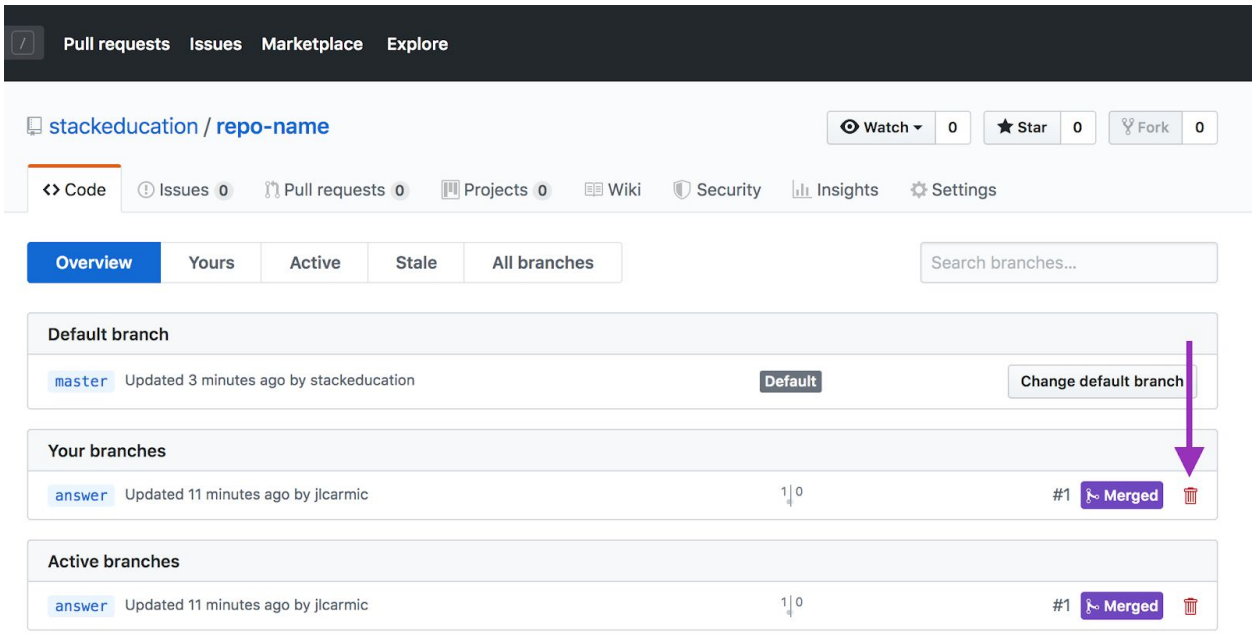
When you are done with your work (typically when your branch has been merged to master) you'll want to clean things up. No need to hang on to old merged branches. Deleting a branch remotely on GitHub can be done with either the delete button that will appear immediately after clicking merge on your PR



or by viewing the branches for the repo.



And clicking on the delete button found there.



To delete a branch locally simply move to a different branch with the checkout command and run the command `git branch -D <BRANCH_NAME>`