2.4.6

Use Git Version Control to Save Our Work

Great job on learning about the GitHub workflow and how it relates to version control. In this step, we will sync up our local branch with the remote branch in our GitHub repository, then merge it into the default branch, `main`.

Before we dive in, let's watch the following video on Git flow:

In this activity, we'll follow the GitHub workflow to update the `main` default branch with the project starter code. In the real world, this would update the remote default branch, `main`, allowing team members to download the starter code.

Expected Behavior

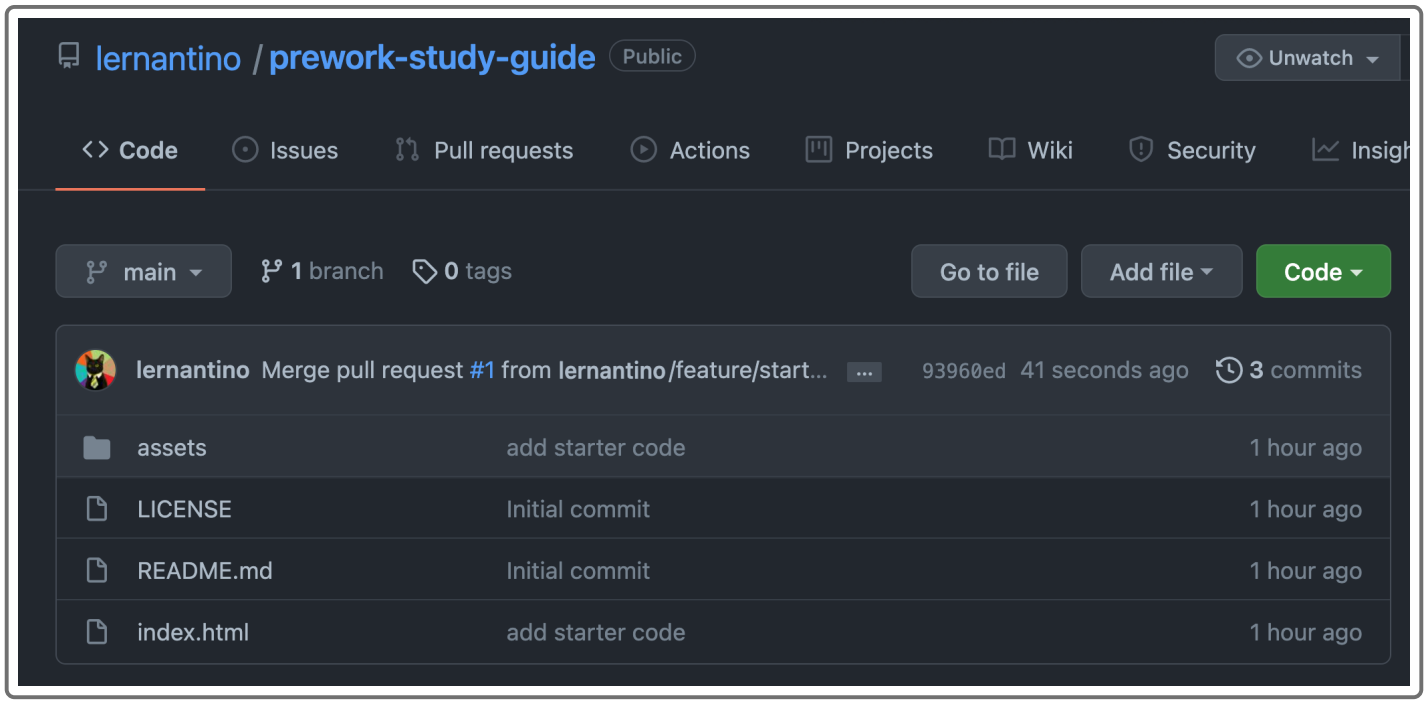
User story:

- As a boot camp student, I want to use Git version control so that I can keep track of the changes I make to my project in my `prework-study-guide` GitHub repository.

Acceptance criteria:

- It is done when I add, commit, and push up the starter code in a feature branch.
- It is done when I open a pull request with my feature branch as the comparison branch and the default branch as the base branch.
- It is done when I merge the PR so that the `main` branch has the starter code for the project.

When we have completed the Git workflow, the GitHub repo default branch will look like the following image:



Let's Code!

To complete this activity, follow these instructions:

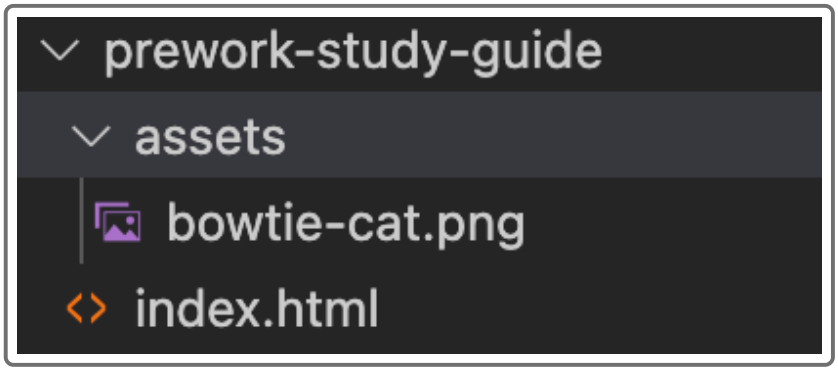
1. In the command line, navigate to the `prework-study-guide` directory. If not currently in this directory, use the following commands from the current user's home directory:

```
cd bootcamp
cd prework-study-guide
```

2. Once inside the `prework-study-guide` directory, open the directory in VS Code if it's not currently open.

```
code .
```

3. In VS Code, check to make sure all of the starter code files for the project are still in there.



4. To check what branch we are currently on, we enter the following git command in the command line:

```
git status
```

a. Right now, we should be in the `main` branch but remember, we don't want to add our work there until it is done and approved. So for now, we need to create a new branch to hold our work.

b. To add a branch where we can save our work-in-progress, we will create a new feature branch.

5. Create a feature branch named “starter-code” by writing the following git command in the command line:

```
git checkout -b feature/starter-code
```

6. Now, let's take a look at what we just did. In the preceding command, a new branch, `feature/starter-code`, is created and the working branch is changed to the new branch.
    - The `git` command must precede every Git statement.
    - The `checkout` command is to move the working branch to a new branch.

- The `-b` flag creates a new branch.
- The `feature/starter-code` is the name of the new branch. We have added the prefix, `feature`, to indicate that it is a feature. It is good practice to name the branches for the feature that will be developed by them to help indicate the purpose of each branch.

7. Now that we have created a new feature branch and made it the current working branch, let's verify this is true with the command `git status` once again. We should now be working in our new feature branch.

```
git status
```

8. To add the starter code to the current working branch, `feature/starter-code`, we'll need to do the following:

   a. First enter the following Git command in the command line:

```
git add -A
```

   b. Take a look at the command we just wrote. The `add` command adds modifications in the current working branch to the staging area. The `-A` flag indicates that we want to add all changes. The staging area informs Git that the changes will be added in the next commit.

   c. Now that we have staged those files to be committed, let's enter the `git commit` command with a commit message that describes what the commit contains. Enter the following git command in the command line:

```
git commit -m "add starter code"
```

   d. Let's take another look at what we just wrote. In the preceding Git statement, a commit was created that captures a snapshot of the currently staged changes. Once committed, `feature/starter-code` will be updated with the starter code.

DEEP DIVE ▼

9. Before we push this up to our repo, we should double-check if our local branch is in sync with the base branch in GitHub. So let's pull down any changes from the base branch.

   a. Enter the following git command in the command line:

```
git pull origin main
```

   b. Let's take a look at what we just wrote. Once again, we identify a Git command with the `git` keyword. Then use the `pull` command to receive a branch's modifications into the local environment. The `origin` keyword indicates the source of the pull will be in the GitHub repo. The `main` keyword indicates the branch. In our case, we are receiving a branch called `main` from the GitHub repository.

   c. It is important to always do a git pull to make sure that you have the latest version of the project on your local computer before you start to work. Otherwise, you might be working on outdated code!

10. Great! Now our local branch has synced with the remote base branch, let's push our changes to the remote branch by using the following command:

```
git push origin feature/starter-code
```

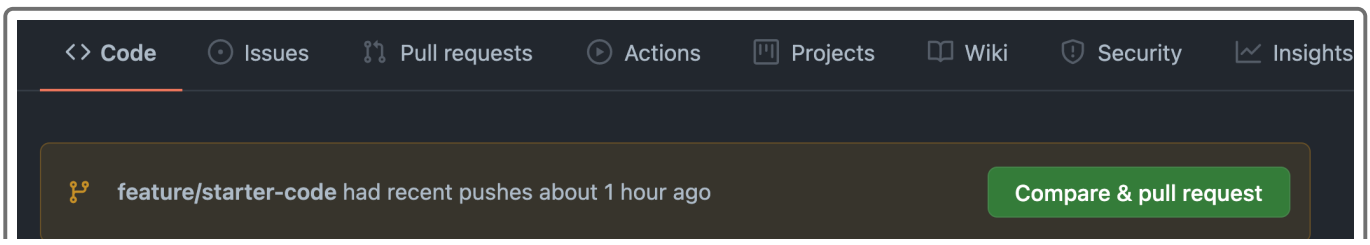
    a. Let's take a look at what we just wrote. In this case, we are using a new command `push` to push the changes we have locally to our remote GitHub branch.

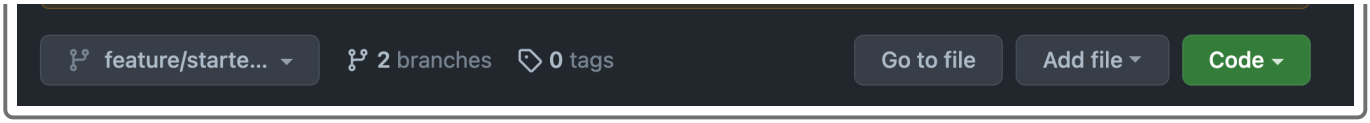
    b. In the browser, navigate to your remote branch on GitHub. As you'll see, there is now a remote `feature/starter-code` branch in GitHub. When using the `push` command, you can create a new remote branch or add new changes to an existing branch.

11. Take a minute to review what we have done so far. We will follow these exact steps every time we work on a project in our boot camp, so it is important to keep practicing and memorize it!

12. Now let's take a closer look at the GitHub repository in our browser.

    a. Go to the GitHub repo in the browser, where the `main` branch is displayed by default. Select the branch `feature/starter-code` from the dropdown. Next, select the "Compare & pull request" button as seen in the following screenshot:

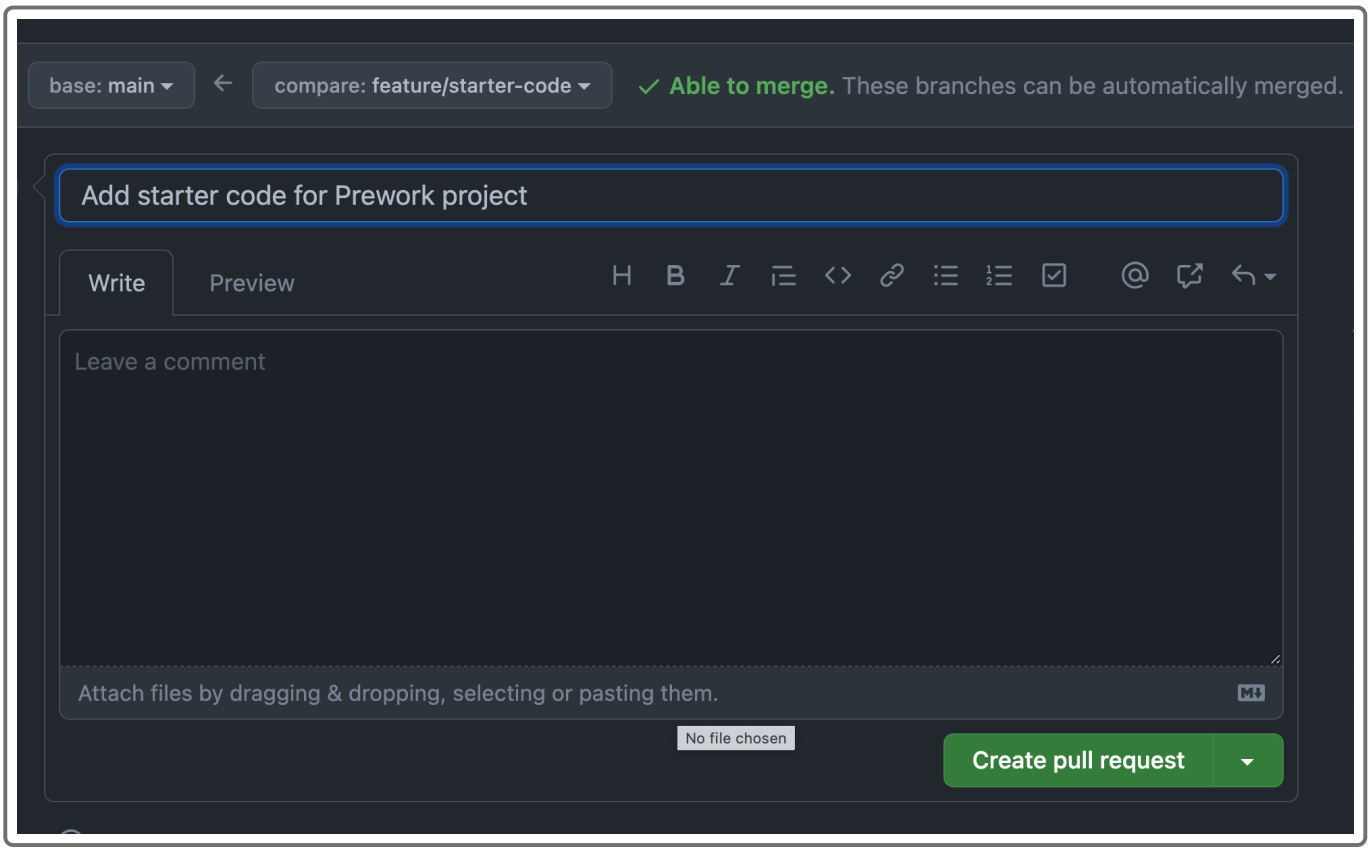




b. Next we'll give the pull request (PR) a descriptive title, like “Add starter code for Prework project”. In the body of the PR, typically we would add instructions for team members that will review the PR about what changes were made, how the feature is supposed to function, and how to start the application so the feature can be tested. For this example, we will not add this content.

c. Since we would like to merge the starter code in the feature branch into the `main` branch, we must designate the base branch as `main` and the comparison branch as `feature/starter-code`.

d. Next, select the "Create Pull Request" button to make your first PR, as shown in the following image:

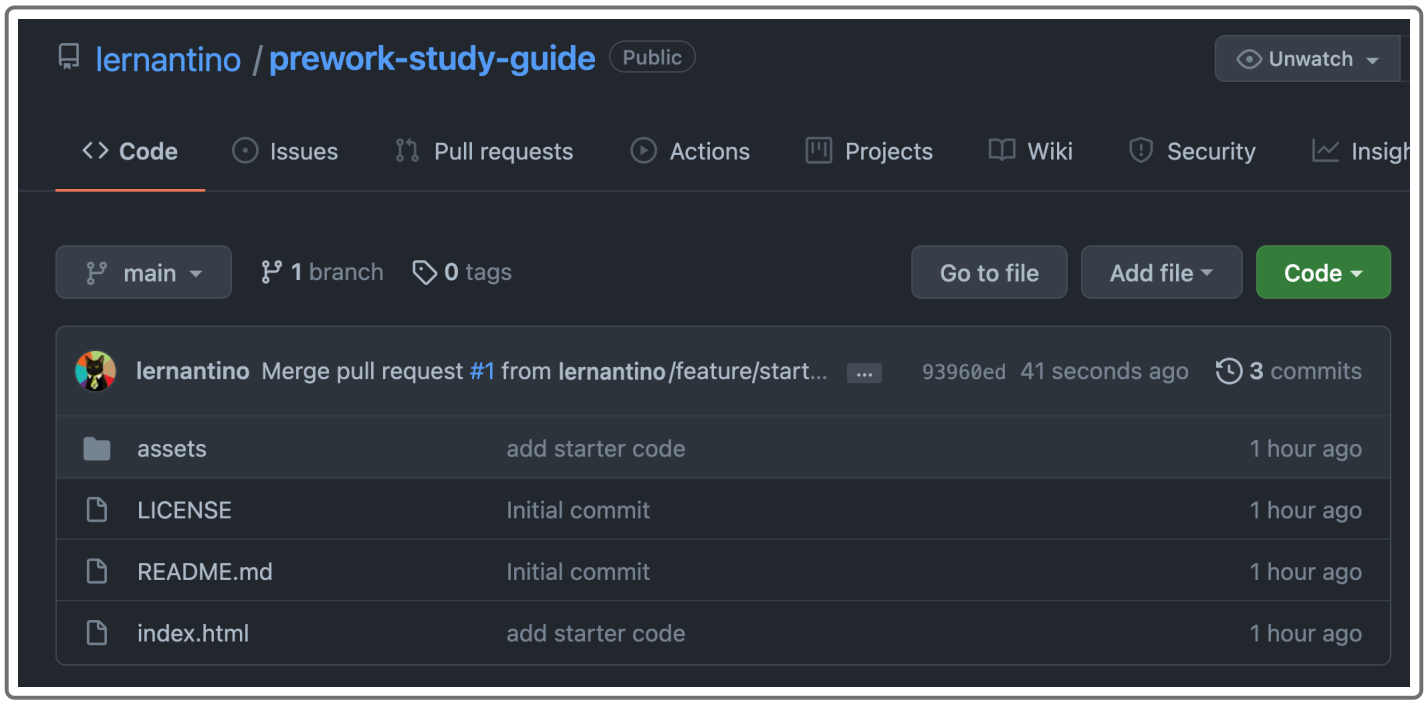


13. In the workplace and when working collaboratively as a team, once a PR has been created, we would request that one or two other team members review the code before it is merged into the base branch. GitHub has a pull request review feature that is a handy quality assurance step necessary to ensure that only completed, quality code is merged into the base branch. But for now, since we are just practicing, we'll merge the PR ourselves by selecting the "Merge Pull Request" button.

Code Review

How did it go? Let's take a moment to review what this might look like.

If everything was done correctly, our `main` branch should now hold our starter code. To check, go to the homepage of the repo and select the `main` branch in the GitHub repo. There, we can that see the starter code is in your repo, as shown in the following image:



We can compare the preceding image with the starter code in our local environment to verify that the remote repo's base branch has been updated with the starter code.

Summary

Great job! Now we know how to keep our local branch in sync with the remote branch.

This will become clearer in the next module when we dive into coding the study guide step by step. But really, the value of version control will make the most sense when you are working on your first group project with multiple collaborators. Until then, it is important to practice an effective Git workflow.