C2S2 Git Tutorial

Johnny Martinez

1 Git Setup

Go to this link to install Git. To start, we need to edit our config file by executing the following commands:

```
git config –global user.name “YOUR NAME”
git config –global user.email EMAIL
git config –global init.defaultBranch “main”
git config –global core.editor “code”
```

Ensure that the email you use is the one tied to your GitHub account. The first two commands configure Git to associate your name and email with your commits. The third command sets your default branch name to “main.” The fourth command sets VSCode as your standard text editor for Git. Now, when you type “code” into the command line, a VSCode window will appear. If you would like to edit your config file directly, type

```
git config –global -e
```

2 Making a Repository

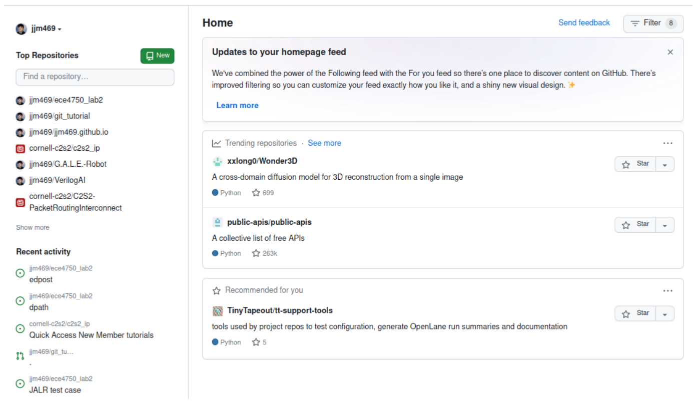
To make a repository, navigate to the directory you would like to work out of, and execute the following commands:

```
mkdir tutorial
cd tutorial
```

Note, as a general rule of thumb, do not put spaces in file or folder names!

Now, let’s make a GitHub repository. First, go to the GitHub website. Click on the green button labeled “new.”

After pressing the button, your screen should look like figure 2. Let’s name our repository “git-tutorial.” We can leave the description blank for now, but just know it is there for future reference. Also, be sure to select ”Add a README file.” Now push the “create repository button.”


jjm469

Top Repositories

jjm469/ece4750_lab2
jjm469/git_tutorial
jjm469/jjm469.github.io
cornell-c2s2/c2s2_ip
jjm469/G.A.L.E.-Robot
jjm469/VerilogAI
cornell-c2s2/C2S2-PacketRoutingInterconnect

Recent activity

jjm469/ece4750_lab2
edpost

jjm469/ece4750_lab2
dpath

cornell-c2s2/c2s2_ip
Quick Access New Member tutorials

jjm469/git_tu…
.

jjm469/ece4750_lab2
JALR test case

Home

Updates to your homepage feed

We've combined the power of the Following feed with the For you feed so there's one place to discover content on GitHub. There's improved filtering so you can customize your feed exactly how you like it, and a shiny new visual design. ✨

Trending repositories · See more

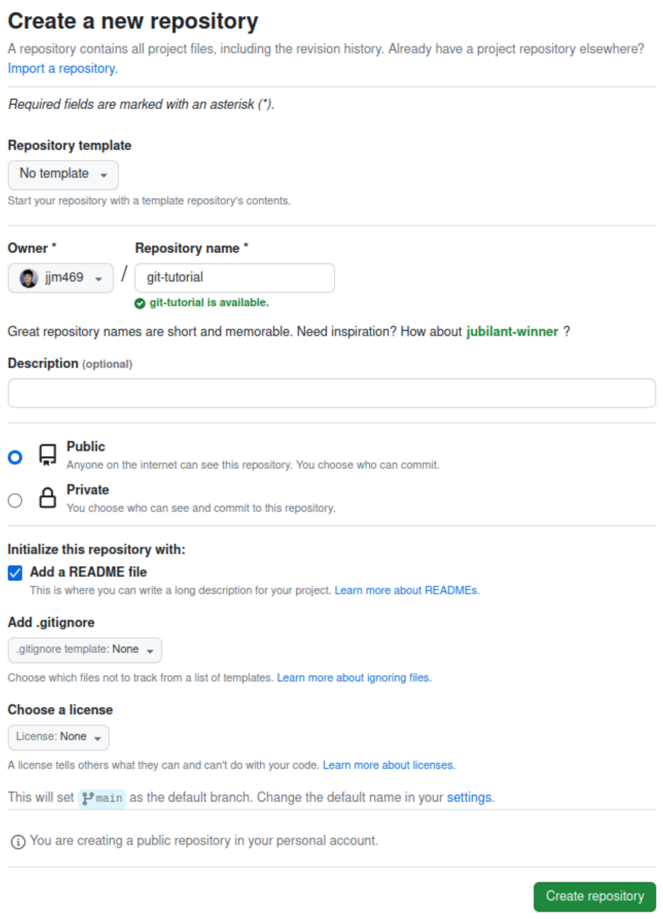
xxlong0/Wonder3D
A cross-domain diffusion model for 3D reconstruction from a single image
● Python ☆ 699

public-apis/public-apis
A collective list of free APIs
● Python ☆ 263k

Recommended for you

TinyTapeout/tt-support-tools
tools used by project repos to test configuration, generate OpenLane run summaries and documentation
● Python ☆ 5

Figure 1: GitHub homepage


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Repository template

No template

Start your repository with a template repository's contents.

Owner * jjm469 / **Repository name *** git-tutorial

✓ git-tutorial is available.

Great repository names are short and memorable. Need inspiration? How about **jubilant-winner** ?

Description (optional)

[✓ **Public**] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

[✓] **Add a README file**
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set `main` as the default branch. Change the default name in your settings.

ⓘ You are creating a public repository in your personal account.

Figure 2: GitHub repository creation page

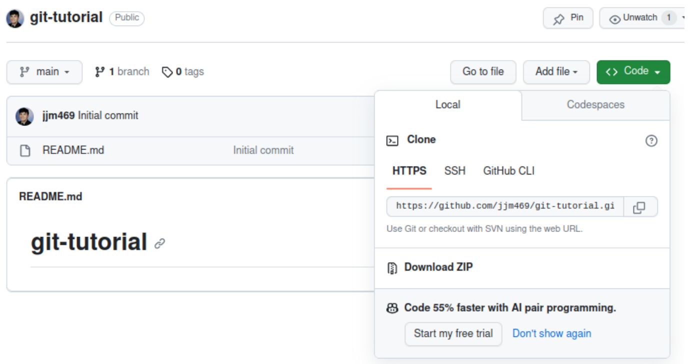


Figure 3: HTTPS link to clone repository

Great! We should now have an empty GitHub repository (aside from the README file). Let's clone the repository into the local directory we created before. To do this, press the green button labeled "code." Copy the link in the HTTPS section.

Now, open the terminal. Make sure you are working inside of the folder you previously made and execute the following:

```
git clone <REPO LINK>
```

If all went well, we have now cloned the remote repository in our local directory! You can check to see if the repository has been cloned by typing "ls" in the command line, and ensuring there is a folder called "git-tutorial" in your current directory. Let's populate our local repository with some files by executing the following:

```
cd git-tutorial
echo "a text file named file 1" >file1.txt
echo "a text file named file 2" >file2.txt
echo "// A blank Verilog file" >module.v
mkdir subdir
cd subdir
echo "a text file named file 3" >file3.txt
cd ..
```

The above commands make two text files (file1, file2), one Verilog file (module), one subdirectory (subdir), and another text file (file3) within subdir. "cd .." is used to navigate to the parent directory of your current directory.

3 The Git Workflow

Before proceeding, I want to take a second to describe the Git Workflow. It is mainly divided into four sections:

1. **Working Directory:** The working directory, also known as the working tree, is your local directory where you create, modify, and organize your project's files. It's the place where you do your actual work.

2. **Staging Area (Index):** The staging area, often referred to as the "index," acts as a middle ground between your working directory and the commit history. It's where you prepare and organize your changes before committing them. When you make changes to files in your working directory, you can choose to stage specific changes or files for the next commit. This selective staging allows you to commit only the changes you want, rather than everything in your working directory.

3. **Commit:** When you commit your changes, you're taking a snapshot of the changes in the staging area and creating a new commit in Git's version history. Each commit contains the changes you've staged. Commits are permanent and represent milestones in your project's history.

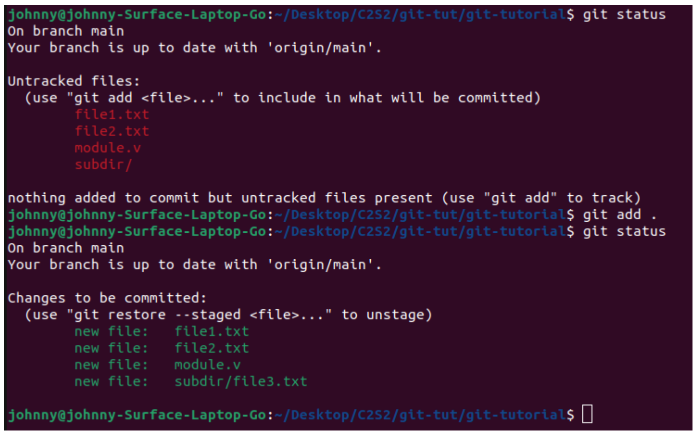
4. **HEAD Commit:** The HEAD is a special pointer that points to the latest commit in the current branch. It represents the snapshot of your project's files as they exist in your local repository at the moment. When you switch branches or checkout a specific commit, the HEAD is updated to reflect the state of the selected branch or commit.

Now that we have established the basic Git Workflow, let's learn how to use it in practice.

3.1 Add

As I mentioned before, we need to actually add our changes in the working directory, and forward them to the staging area. To do this, we used the git add command. Observe the following commands:

```
git add file1.txt
git add file2.txt
git add module.v
git add subdir/file3.txt

git add *.txt
git add *.v

git add .
```



```
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
        module.v
        subdir/

nothing added to commit but untracked files present (use "git add" to track)
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git add .
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   module.v
        new file:   subdir/file3.txt

johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$
```

Figure 4: Git status before and after adding files to staging area

Above, I have listed three possible methods to add our files to the staging area (Each method is separated by a blank line and I will refer to them as methods 1, 2 and 3 respectively. In method 1, we are individually adding all of the files that we want to stage. In method 2, we make use of a wildcard character (*) to stage all text files and verilog files. In method 3, we are recursively staging all files within the working directory (this is the most common way to add things to the staging area). To see what files are in the staging area, execute the following command:

```
git status
```

In the terminal, you will be shown "untracked files" or the files not on the staging area, "changes to be committed" or the files on the staging area, and commits. Right now, you should have no commits. If you would like to remove something from the staging area, execute one of the following:

```
git restore –staged <FILE>

git restore –staged .
```

The first option is useful for removing individual files, while the other clears the whole staging area. You can execute "git status" again to check that the file you specified has been unstaged. To clear the whole staging area, execute:

3.2 Commit

Now that we have added all of our files to the staging area, we want to save them. To do this, run the following command:

```
git commit -m “initial commit”
```

By executing this command, we are essentially taking a snapshot of the staging area, and saving it. Notice, we use the “-m” flag to leave a commit message. It is good practice to leave a commit message describing what has been changed or fixed. If you do not include the -m flag in your commit command, a text file will open up for you to write your commit message.

Commit Best Practices

1. **Commit Early and Often:** Make small, focused commits as you work on your project. This allows you to track changes incrementally and makes it easier to identify when and where issues were introduced.

2. **Atomic Commits:** Each commit should address a single, logical change or feature. This practice keeps your commit history clean and makes it easier to manage and review changes.

3. **Commit Messages Matter:** As I mentioned before, commit messages are important! Take a moment to think about your commit messages. Explain not only ”what” was changed but also ”why” it was changed. Clear and informative commit messages are essential for effective collaboration.

The reason we want to frequently commit, and leave descriptive messages is in the case we need to revert our code. Having descriptive messages on the state of your code for each snapshot will be useful in this situation. We will go further into depth on reverting to a previous commit in a later section, but for now just keep this in mind.

3.3 Push

Now that we have committed our local changes, we want to push them to the remote repository. To do so, type:

```
git push
```

Now, if you view your repository on the GitHub website, you should see that your repository has more than just a README file! As a matter of fact, it now should have all of the files that were present in our working directory.

4 Branches

Recall, Git stores your code in a series of commits. The most recent commit is the head commit. When you create a branch, instead of adding a commit to your main branch or the existing chain of commits, you initiate a distinct series of commits that diverges or branches off from the main branch pointed to by

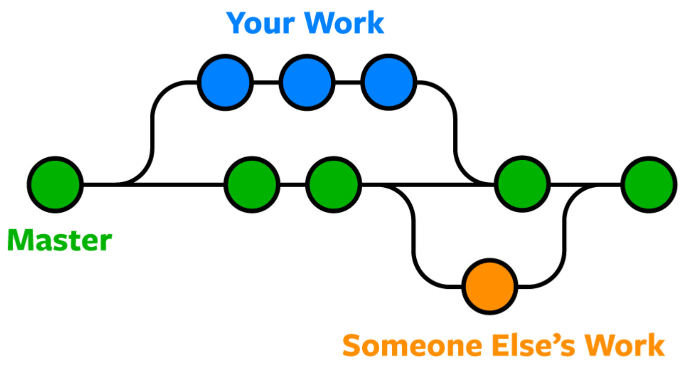


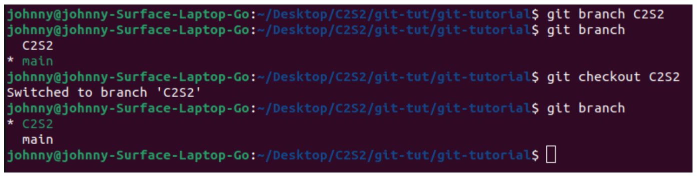
Figure 5: Branch visualization

the head commit. This branch allows you to work on new features, bug fixes, or experiments independently without affecting the main branch until you're ready to merge your changes back into it. To make a branch, execute the following command:

---

git branch <YOUR BRANCH NAME >

Let's name our branch "c2s2" for this tutorial. To see all of your branches, run "git branch" without specifying a branch name. You should see a list like this in the command line.



```
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git branch C2S2
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git branch
  C2S2
* main
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git checkout C2S2
Switched to branch 'C2S2'
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git branch
* C2S2
  main
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ 
```

Figure 6: Enter Caption

The asterisk indicated the branch you are currently working on. To switch to the branch we just created, execute:

---

git checkout "C2S2"

If you run "git branch" again, your terminal should look something like this. Notice, the asterisk is now on the "c2s2" branch. Let's populate the branch with some files.

---

echo >branch1.txt
echo >branch2.txt

Now, we can commit and push the changes to our new branch as we would if on the main branch. The one change we make is that we replace “main” with the name of our branch (in this case “c2s2”) in the push command.

```
git add .
git commit -m“added branch1, branch2 text files.”
git push -u origin C2S2
```

Here, you need “-u origin c2s2” in the push command because git doesn’t have the remote branch linked to yours yet. A couple of clarifying notes: “origin” is the default alias for the remote repository we are pushing to. The branch name ”c2s2” is the new name for the remote branch you want to create.

If you now go to the GitHub website, and view the repository, you should see a green button at the top of the screen labeled “Compare and pull request.” Before we cover pull requests, I want to clarify one thing. If you clone a directory locally, and run “git branch” you may not see all of the branches available in your local repository. To view all of the branches of your repository, run

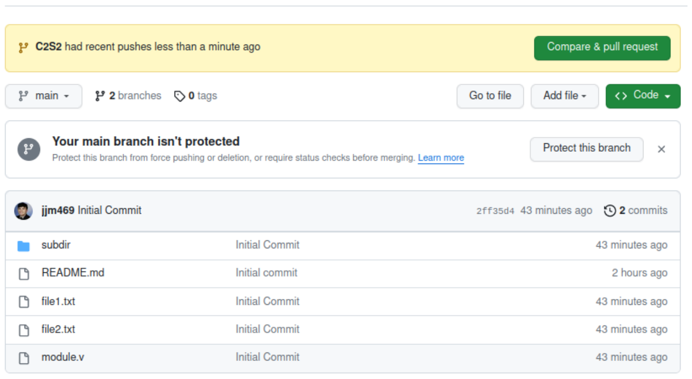
```
git branch -a
```

And, if you want to switch to one of these remote branches, just use ”git Checkout.” Furthermore, if another person working on the same repository makes a new branch, to access it, you must run

```
git fetch –all
```

5 Pull Requests

Recall, after we pushed our branch to the remote repository, and opened GitHub, we were prompted with a button to create a pull request.


C2S2 had recent pushes less than a minute ago

Compare & pull request

main · **2** branches · **0** tags · Go to file · Add file · Code

Your main branch isn't protected

Protect this branch from force pushing or deletion, or require status checks before merging. Learn more

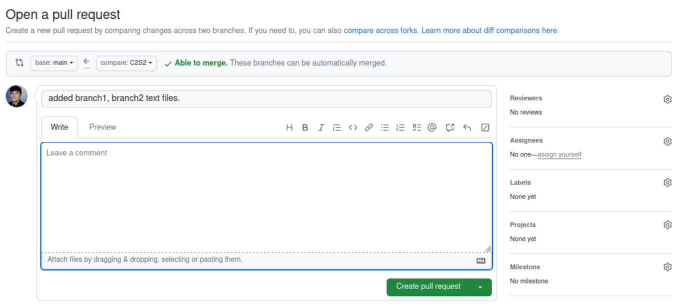
Protect this branch

jjm469 Initial Commit · 2ff35d4 43 minutes ago · **2** commits

subdir	Initial Commit	43 minutes ago
README.md	Initial commit	2 hours ago
file1.txt	Initial Commit	43 minutes ago
file2.txt	Initial Commit	43 minutes ago
module.v	Initial Commit	43 minutes ago

Figure 7: GitHub pull request prompt 1

To merge the "C2S2" branch with our main branch, we need to open a pull request. To do so, press the button prompting us to open a pull request. Your screen should now look like figure 8.


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: main ← compare: C2S2 ✓ **Able to merge.** These branches can be automatically merged.

added branch1, branch2 text files.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Reviewers: No reviews

Assignees: No one—assign yourself

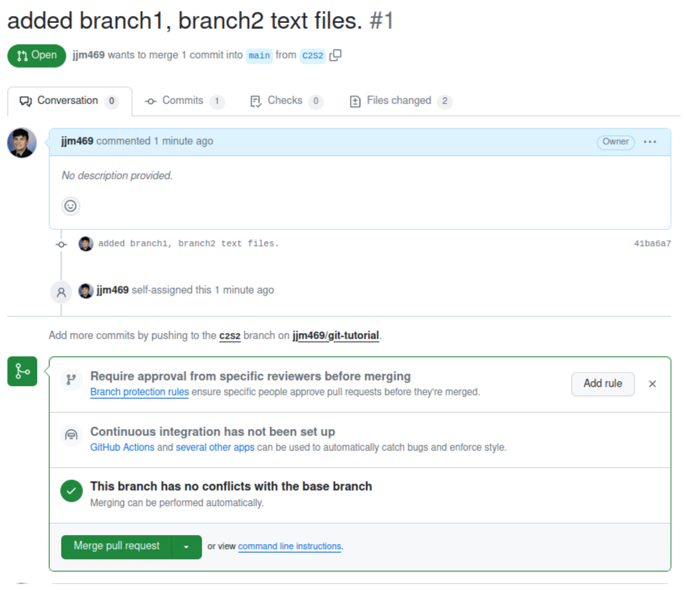
Labels: None yet

Projects: None yet

Milestone: No milestone

Figure 8: GitHub pull request prompt 2

If you look at figure 8, you will see on the right there is a section labeled "assignees," as well as a button prompting you to assign yourself. An assignee is someone tasked with working on a specific pull request or issue. A reviewer is someone tasked with reviewing, and approving a pull request; you cannot assign yourself as a reviewer of your own PR so it's impossible to show that in a tutorial, but just know it exists. Let's assign ourselves to the pull request. To open our pull request, write a description of the pull request in the text box, and press the button labeled "Create pull request."



added branch1, branch2 text files. #1

Open jjm469 wants to merge 1 commit into `main` from `c2s2`

jjm469 commented 1 minute ago — Owner

No description provided.

`added branch1, branch2 text files.` `41ba6a7`

jjm469 self-assigned this 1 minute ago

Add more commits by pushing to the **c2s2** branch on **jjm469/git-tutorial**.

Require approval from specific reviewers before merging
Branch protection rules ensure specific people approve pull requests before they're merged.

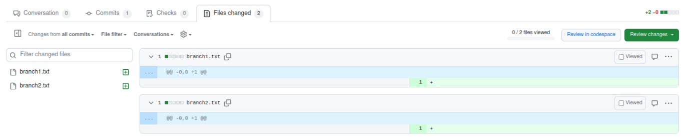
Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

or view command line instructions.

Figure 9: GitHub pull request prompt 3

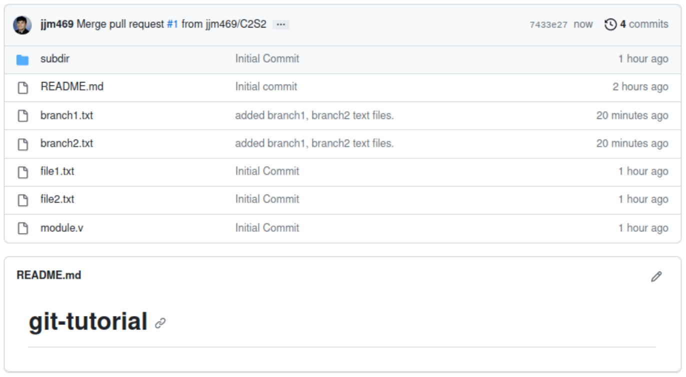
If you want to view the changes that have been made to any files in the pull request, go to the ”Files changed” section. Because the files we added were blank text tiles, this section is pretty empty right now. Typically, there will be more insertions and deletions that happen, and this section will be more densely populated.



```
1 branch1.txt
@@ -0,0 +1 @@
1 +

1 branch2.txt
@@ -0,0 +1 @@
1 +
```

Figure 10: Files changed section

Notice, our branch has no merge conflicts and can be merged. A merge conflict occurs when there are conflicting changes in different branches, and Git is unable to automatically resolve the differences during a merge. This typically happens when two or more contributors make changes to the same part of a file or to related lines of code in their respective branches. Merge conflicts can also arise when a file is deleted in one branch but modified in another. We will go over resolving merge conflicts in a later section. For now, let’s merge our branch by pressing the button labeled “Merge pull request.”



jjm469 Merge pull request #1 from jjm469/C2S2 ··· 7433e27 now 4 commits

subdir	Initial Commit	1 hour ago
README.md	Initial commit	2 hours ago
branch1.txt	added branch1, branch2 text files.	20 minutes ago
branch2.txt	added branch1, branch2 text files.	20 minutes ago
file1.txt	Initial Commit	1 hour ago
file2.txt	Initial Commit	1 hour ago
module.v	Initial Commit	1 hour ago

README.md

git-tutorial

Figure 11: Updated repository

Now, if you view the code section, you will notice our main branch has the branch1 and branch2 text files we created. We now also need to ensure that our local repository is up to date. To do this, run the following

```
git checkout main
git pull
```

The “git pull” command is a convenient way to update your local branch with the latest changes from a remote repository in a single step. It helps keep your local branch in sync with the remote branch, making it useful for collaborative and team-based development. However, it’s essential to watch for merge conflicts if there are conflicting changes between your local branch and the remote branch. Otherwise, you will not be able to pull from the remote repository.

Sometimes, you are not prompted with the button to create a pull request. In this case, you need to go to the section labeled ”Pull requests.” Here, not only can you open a new pull request, but you can see other opened pull requests made by either yourself or your team members.

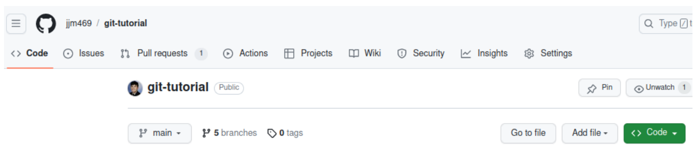


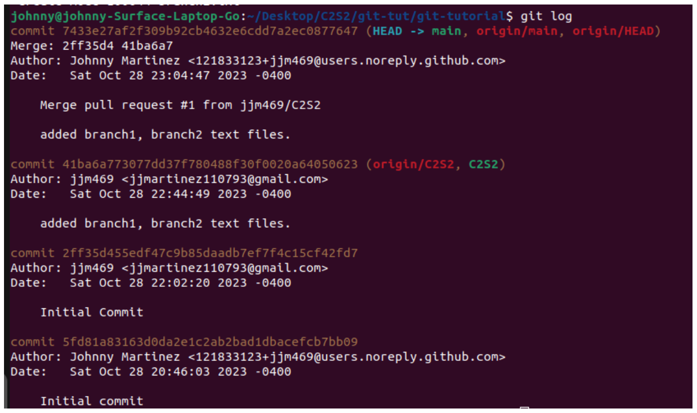
Figure 12: Pull requests section

6 Reverting Commits

As I am sure you know by this point, Git stores code in a series of commits. To revert a commit, you can execute the following command:

```
git checkout <COMMIT HASH>
```

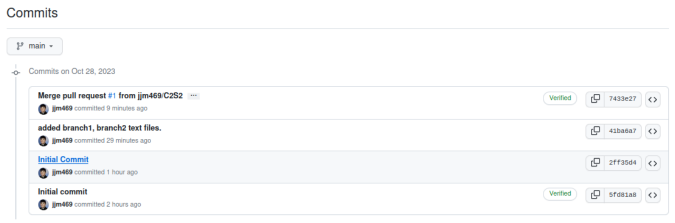
There are multiple ways to access commit hashes. The first way is in your terminal. By running the "git log" command, you can view all commits in the history of the repository. The second way is via the GitHub website.



```
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git-tut/git-tutorial$ git log
commit 7433e27af2f309b92cb4632e6cdd7a2ec0877647 (HEAD -> main, origin/main, origin/HEAD)
Merge: 2ff35d4 41ba6a7
Author: Johnny Martinez <121833123+jjm469@users.noreply.github.com>
Date:   Sat Oct 28 23:04:47 2023 -0400

    Merge pull request #1 from jjm469/C2S2

    added branch1, branch2 text files.

commit 41ba6a773077dd37f780488f30f0020a64050623 (origin/C2S2, C2S2)
Author: jjm469 <jjmartinez110793@gmail.com>
Date:   Sat Oct 28 22:44:49 2023 -0400

    added branch1, branch2 text files.

commit 2ff35d455edf47c9b85daadb7ef7f4c15cf42fd7
Author: jjm469 <jjmartinez110793@gmail.com>
Date:   Sat Oct 28 22:02:20 2023 -0400

    Initial Commit

commit 5fd81a83163d0da2e1c2ab2bad1dbacefcb7bb09
Author: Johnny Martinez <121833123+jjm469@users.noreply.github.com>
Date:   Sat Oct 28 20:46:03 2023 -0400

    Initial commit
```

Figure 13: Git log

Notice, in figure 11, at the top right of the picture there is a clock icon followed by the number of commits. Click on it. Now, you have access to all of the commit hashes and can use them to restore past commits.



Commits

main

Commits on Oct 28, 2023

Merge pull request #1 from jjm469/C2S2 — jjm469 committed 9 minutes ago — Verified — 7433e27

added branch1, branch2 text files. — jjm469 committed 29 minutes ago — 41ba6a7

Initial Commit — jjm469 committed 1 hour ago — 2ff35d4

Initial commit — jjm469 committed 2 hours ago — Verified — 5fd81a8

Figure 14: Commit hash