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 <u>GitHub</u> website. Click on the green button labeled "new."

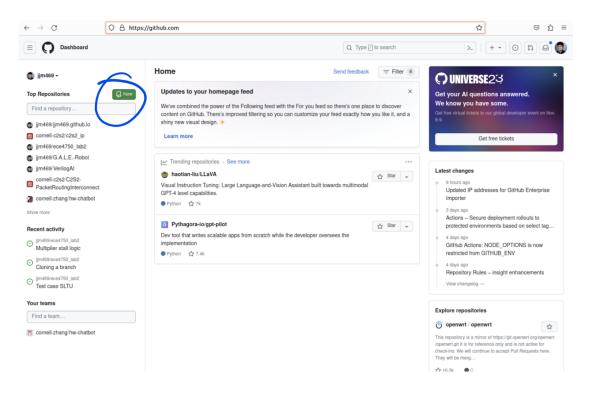


Figure 1: GitHub homepage

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, make sure "Add a README file" is selected. Now push the "create repository button."

$\leftarrow \rightarrow G$	O A https://github.com/new	90% 🏠	ල එ =
	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 Surry repository as template repository's contents.		
	Owner * Repository name *		
	Great repository names are short and memorable. Need inspiration? How about fuzzy-parakeet ? Description (optional)		
	Public Anyone on the internet can see this repository. You choose who can commit.		
	Algorithm of the finite many care set or the regulatory in the Calculator who can care and convert to this repeatory.		
	Initialize this repository with: 2 Add a README file This is where you can write a long description for your project. Learn more about READMEs.		
	Add upblignore galgione template. The Choose which files not to track from a list of templates. Laws more about ignoring files.		
	Choose a license		
	A iscense tells offers what they can and can't do with your code, Learn more about iscenses. This will set Pressin as the default banch. Change the default name in your settings .		
	You are creating a public repository in your personal account. You are creating a public repository in your personal account.		
	Create repository	-	

Figure 2: GitHub repository creation page

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.

$\leftarrow \rightarrow C$ O A https://github.com/jjm469/git_tutorial		90% රූ 🗢 ව් =
☐ Ø jjm469 / git_tutorial ◇ Code ⊙ Issues th Pull requests ⊙ Actions ⊞ Projects ⊞ Wiki ⊙ Security	Q Type [/]to search ≻ (+ •) ⊙ 11 @ ④
git_tutorial (Public)	☆ Pin	I ▼ V Fork 0 ▼ ☆ Star 0 ▼
12 main - 12 to tranch So tags	Go to file Add file - <> Code -	About 🛞
ijm469 Initial commit	Local Codespaces (New)	No description, website, or topics provided.
C README.md Initial commit	Clone ⑦ HTTPS SSH GitHub CLI	□ Readme - Activity
README.md	https://github.com/jjm469/git_tutorial.gi	 ✿ 0 stars ④ 1 watching
git_tutorial ∉	Use Git or checkout with SVN using the web URL.	V 0 forks
	 Download ZIP Code 55% faster with Al pair programming. 	Releases
	Start my free trial Don't show again	No releases published Create a new release
		Packages
		No packages published Publish your first package

Figure 3: HTTPS link to clone repository

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

git clone <HTTPS 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.

2.a 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, lets learn how to work with it in practice.

2.b Add

As I mentioned before, we need to actually add out 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 .

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 the following:

git restore --staged <file>

You can execute "git status" again to check that the file you specified has been unstaged. To clear the whole staging area, execute:

git restore --staged .

2.c 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

<u>Commit Early and Often:</u> 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.

<u>Atomic Commits:</u> 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.

<u>Commit Messages Matter:</u> 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.

2.d Push

Now that we have saved a snapshot of our local working directory, we want to push it to the remote repository. To do so, type

git push origin main

Note, sometimes you may need to replace "main" with "master".

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. A couple of clarifying notes: "origin" is the default alias for the remote repository we are pushing to. The branch name "main" is commonly used as the default branch name in many git repositories. However, some repositories may use "master" instead of "main" as the default branch name. Generally, it is not good practice to push directly to main. We circumvent this issue by creating branches and merging them with "main." This is a perfect segway into our next section which is about branches!

3.) Branches

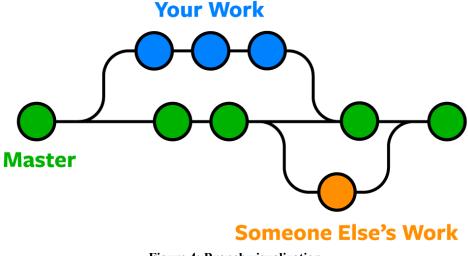


Figure 4: Branch visualization

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 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_tutorial$ git branch c2s2
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git_tutorial$ git branch
    c2s2
* main
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git_tutorial$ []
```

Figure 5: Command line snapshot 1 showcasing selected branch

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.

```
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git_tutorial$ git checkout c2s2
Switched to branch 'c2s2'
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git_tutorial$ git branch
* c2s2
main
johnny@johnny-Surface-Laptop-Go:~/Desktop/C2S2/git_tutorial$
```

Figure 6: Command line snapshot 2 showcasing changed selected 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 origin c2s2

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 & pull request." Before we cover pull requests, I want to explain a couple of other features of branches. If you want to clone a specific branch directly, execute:

git clone -b

branchname> <remote-repo-url>

Furthermore, 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 any branch, use "git checkout."

4.) 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.

	69 / git_tutorial	Q Type 🛛	to search	- + • • • n 🗠 🚳
\leftrightarrow Code \odot	ssues 🖏 Pull requests 💿 Actions 🖽 Projects 🖽 Wiki 🔃 Security 🗠 Ir	nsights tôj Settings		
	git_tutorial (Public)	Pin SUnwatch 1	▼ V Fork 0 v C Star	• •
	p c2s2 had recent pushes 1 minute ago	Compare & pull request	About No description, website, or topics	\$ ³
	p main → p 2 branches ⊘0 tags	Go to file Add file - <> Code -	provided.	
	Pour main branch isn't protected Protect this branch from force pushing or deletion, or require status checks before merging.	Protect this branch ×	 小 Activity ☆ 0 stars ⊙ 1 watching 	
	jim469 initial commit	7724233 last week 3 2 commits	약 0 forks	
	subdir initial commit	last week	Releases	
	README.md Initial commit	last week	No releases published	
	file1.txt initial commit	last week	Create a new release	
	🗅 file2.txt initial commit	last week		
	module.v initial commit	last week	Packages	
	README.md	1	No packages published Publish your first package	
	git_tutorial ∉		Languages Verilog 100.0%	
	O © 2023 GitHub, Inc. Terms Privacy Security Status Docs	Contact GitHub Pricing API Training	Blog About	

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 green button.

≡ () jjm469 / git	t_tutorial			Q Type 🖉	to search	>_ + • O n @ 🥘
<> Code	្រុ Pull re	quests 🕞 Actions 🗄 Projects 🖽 Wiki	🛈 Security 🗠 Insights 🕸 Settings			
	Create a		nches. If you need to, you can also compare across forks. Learn m	iore about diff c	omparisons here.	
		added branch1, branch2 test files			Reviewers No reviews	ŵ
		Write Preview Leave a comment	нв <i>і</i> і ⇒	2 ~ 2	Assignees No one— <u>assign yourself</u>	鐐
					Labels None yet	\$
		Attach files by dragging & dropping, selecting or pasting	Rom		Projects None yet	\$
		Attach mes by uragging a uropping, selecting or pasing	Create pull requ	uest -	Milestone No milestone	\$
		Remember, contributions to this repository should follow ou	r <u>GilHub Community Guidelines</u> .		Development Use <u>Closing keywords</u> in the descriptic automatically close issues	on to
					Helpful resources GitHub Community Guidelines	
		-o- 1 commit	1 2 files changed		At 1 contributor	
	-0- Co	mmits on Oct 19, 2023				
		added branch1, branch2 test files			L bes	85834

Once you do that, your screen should look like figure 8 shown below.

Figure 8: Figure 7: GitHub pull request prompt 2

If you scroll down on the screen, you can also see a section like the one below, outlining added files, changes to pre existing files, etc. 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.

added branch1, branch2 test files	L be85834
ijm469 committed 53 minutes ago	
Showing 2 changed files with 2 additions and 0 deletions.	Split Unifie
✓ 1 ■0000 branch1.txt □	
··· ··· 00 -0,0 +1 00	
1 +	
1 + ∨ 1 ■0000 branch2.txt []	

Figure 9: Visualization of additions and deletions to be implemented

If you look at figure 7, 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 reviewing, and in

some cases confirming a pull request. Let's assign ourselves. 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 test files #1	Edit <> Code •
្រា Open j	n469 wants to merge 1 commit into main from c2s2 []	
रू) Conversa	tion 0 -o- Commits 1 F. Checks 0 🗄 Files changed 2	+2 -0
jjm4	69 commented now Owner ····	Reviewers 🕄
Add	ed two text files to our repository.	No reviews Still in progress? Convert to draft
		Assignees 203
-0-	added branch1, branch2 test files be85834	 jjm469
R	jim469 self-assigned this now	Labels 💱 None yet
Add m	ore commits by pushing to the <u>c2s2</u> branch on jjm469/git_tutorial.	Projects 🛱
4	Require approval from specific reviewers before merging Branch protection rules ensure specific people approve pull requests before they're merged.	None yet
		Milestone 🕄
ē	Continuous integration has not been set up GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.	Development 23
Q	This branch has no conflicts with the base branch Merging can be performed automatically.	Successfully merging this pull request may close these issues.
	erge pull request - or view command line instructions.	Notifications Customize
🕘 🛛 w	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	You're receiving notifications because you're watching this repository.
Lea	ve a comment	1 participant
Atta	A files by dragging & dropping, selecting or pasting them.	Cock conversation
	Comment Comment	

Figure 10: GitHub pull request prompt 3

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." On C2S2, we generally do not allow creators of a pull request to confirm their own pull request. Furthermore, we use GitHub Actions to run test cases and ensure modifications to files did not break anything.

) g	it_tutorial Public		S Pin Outwatch	1 ▼ ¥ Fork 0 ▼ ☆ Star 0 ▼	
۲	main - 🖓 2 branches 🕟 0 tage	3	Go to file Add file - <> Code -		
۲	Your main branch isn't prote Protect this branch from force pushing or	No description, website, or topics provided.			
ijm469 Merge pull request #1 from ijm469/c2s2		854f476 1 minute ago 🔞 4 commits	A contra		
	subdir	initial commit	last week	 ⊙ 1 watching ♀ 0 forks 	
ľ	README.md	Initial commit	last week	a o lona	
Ľ	branch1.txt	added branch1, branch2 test files	1 hour ago	Releases	
ľ	branch2.txt	added branch1, branch2 test files	1 hour ago	No releases published	
ľ	file1.txt	initial commit	last week	Oracle a service lance	
ľ	file2.txt	initial commit	last week		
ľ	module.v	initial commit	last week	Packages	
REA	ADME.md		0	No packages published Publish your first package	
ç	git_tutorial ∂			Languages	
				 Verilog 100.0% 	

Figure 11: Updated main branch after successful merge

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.

5.) 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 find 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.

ំ main 👻 វ្រំ 9 branches Ϛ	▶ 0 tags	Go to file Add file ▼ <> Code ▼	About	
	ushing or deletion, or require status checks before merging. <u>Learn</u>		No description, website, or topics provide □ Readme ^r Activity ☆ 0 stars	
jjm469 Merge pull request #6	i from jjm469/change2	65b411f yestelday ③14 commits	 ⊙ 1 watching ♀ 0 forks 	
branch1.txt	added branch1, branch2 test files	yesterday		
branch2.txt	added branch1, branch2 test files	yesterday	Releases	
) change	changed	yesterday	No releases published Create a new release	
) change.txt	changed	yesterday		
j file1.txt	changed file1.txt	yesterday	Packages	
file2.txt	initial commit	2 weeks ago	No packages published Publish your first package	
) module.v	initial commit	2 weeks ago		
new_feature.txt	added new feature	yesterday	Languages	
EADME.md		0	• Verilog 100.0%	

Figure 11: GitHub commit information

By pressing the circled button, you will be brought to a screen with all of your commits. You will have the option to copy a commit hash for any given commit.



Figure 11: GitHub hashes

After you revert your commit, you will be able to make a branch, and make any changes as needed.