

Git Tutorial for beginners

Adapted from An Intro to Git and GitHub for Beginners (Tutorial)
(hubspot.com)

You will learn in this tutorial:

- Git and GitHub
- Remote and local in Git
- Branches
- How to make changes and get changes from a repository
- Get started with GitHub desktop a GUI environment for GitHub

New to git? Follow the steps below to get comfortable making changes to the code base, opening up a pull request (PR), and merging code into the primary branch. Any important git and GitHub terms are in bold with links to the official git reference materials.

Step 0: Install git and create a GitHub account

The first two things you'll want to do are install git and create a free GitHub account. **Software needed:**

- [Git - Downloads \(git-scm.com\)](https://git-scm.com)
- [GitHub Desktop | Simple collaboration from your desktop](#)

Git and GitHub

A quick aside: git and GitHub are **not** the same thing. Git is an open-source, version control tool created in 2005 by developers working on the Linux operating system; GitHub is a company founded in 2008 that makes tools which integrate with git. You do not need GitHub to use git, but you cannot use GitHub without using git. There are many other alternatives to GitHub, such as GitLab, BitBucket, and “host-your-own” solutions such as gogs and gittea. All of these are referred to in git-speak as “remotes”, and all are completely optional. You do not need to use a remote to use git, but it will make sharing your code with others easier.

Step 1: Create a local git repository

When creating a new project on your local machine using git, you'll first create a new [repository](#) (or often, 'repo', for short).

To use git we'll be using the terminal. If you don't have much experience with the terminal and basic commands, [check out this tutorial](#) (If you don't want/ need a short history lesson, skip to step three.)

To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command. For example, if you have a 'projects' folder on your desktop, you'd do something like:

Open command propt and create a new folder MyFirstRepo

```
C:\Users\Do Quoc Binh>mkdir MyFirstRepo  
C:\Users\Do Quoc Binh>cd MyFirstRepo  
C:\Users\Do Quoc Binh\MyFirstRepo>
```

Copy a file into this folder. In this tutorial a file name Mongoose 101.docx has been added

To initialize a git repository in the root of the folder, run the [git init](#) command:

```
C:\Users\Do Quoc Binh\MyFirstRepo>dir /b  
Mongoose 101.docx  
  
C:\Users\Do Quoc Binh\MyFirstRepo>git init  
Initialized empty Git repository in C:/Users/Do Quoc Binh/MyFirstRepo/.git/  
  
C:\Users\Do Quoc Binh\MyFirstRepo>
```

Notice: dir /b -> to view the content of current directory

Step 2: Add a new file to the repo

Once you've added or modified files in a folder containing a git repo, git will notice that the file exists inside the repo. But, git won't track the file unless you explicitly tell it to. Git only saves/manages changes to files that it *tracks*, so we'll need to send a command to confirm that yes, we want git to track our new file.

After creating the new file, you can use the [git status](#) command to see which files git knows exist.

```
C:\Users\Do Quoc Binh\MyFirstRepo>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Mongoose 101.docx

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\Do Quoc Binh\MyFirstRepo>
```

What this basically says is, "Hey, we noticed you created a new file called Mongoose 101.docx, but unless you use the 'git add' command we aren't going to do anything with it."

An interlude: The staging environment, the commit, and you

One of the most confusing parts when you're first learning git is the concept of the staging environment and how it relates to a commit.

A **commit** is a record of what changes you have made since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those changes into a commit.

Commits make up the essence of your project and allow you to jump to the state of a project at any other commit.

So, how do you tell git which files to put into a commit? This is where the **staging environment or index** come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the **git add** <filename> command (see Step 3 below).

Once you've used the git add command to add all the files you want to the staging environment, you can then tell git to package them into a commit using the **git commit** command.

Note: The staging environment, also called 'staging', is the new preferred term for this, but you can also see it referred to as the 'index'.

Step 3: Add a file to the staging environment

Add a file to the staging environment using the git add command.

```
C:\Users\Do Quoc Binh\MyFirstRepo>git add .

C:\Users\Do Quoc Binh\MyFirstRepo>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Mongoose 101.docx

C:\Users\Do Quoc Binh\MyFirstRepo>
```

To reiterate, the file has **not** yet been added to a commit, but it's about to be.

Step 4: Create a commit

It's time to create your first commit!

Run the command `git commit -m "Your message about the commit"`

```
C:\Users\Do Quoc Binh\MyFirstRepo>git commit -m "Your message about the commit"
[master (root-commit) f2ff1f5] Your message about the commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Mongoose 101.docx

C:\Users\Do Quoc Binh\MyFirstRepo>
```

The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. Don't put a message like "asdfadsf" or "foobar". That makes the other people who see your commit sad. Very, very, sad. Commits live forever in a repository (technically you *can* delete them if you really, really need to but it's messy), so if you leave a clear explanation of your changes it can be extremely helpful for future programmers (perhaps future you!) who are trying to figure out why some change was made years later.

Step 5: Create a new branch

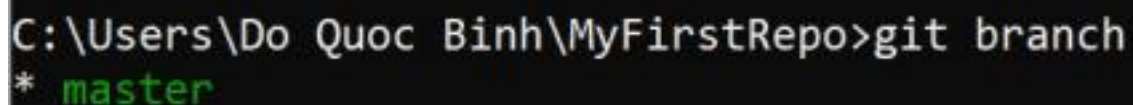
Now that you've made a new commit, let's try something a little more advanced.

Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where [git branches](#) come in.

Branches allow you to move back and forth between 'states' of a project. Official git docs describe branches this way: ‘A branch in Git is simply a lightweight movable pointer to one of these commits.’ For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can [merge](#) your changes from your branch into the primary branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

Let's say you are on the primary branch and want to create a new branch to develop your web page. Here's what you'll do: Run [git checkout -b <my branch name>](#). This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the primary branch.

After running the above command, you can use the [git branch](#) command to confirm that your branch was created:



```
C:\Users\Do Quoc Binh\MyFirstRepo>git branch
* master
```

A terminal window with a black background and white text. The prompt is 'C:\Users\Do Quoc Binh\MyFirstRepo>'. The command 'git branch' has been entered. The output is '* master', where the asterisk is green and 'master' is white.

The branch name with the asterisk next to it indicates which branch you're on at that given time.

[A note on branch names](#)

By default, every git repository's first branch is named 'master' (and is typically used as the primary branch in the project). As part of the tech industry's general anti racism work, some groups have begun to use alternate names for the default branch (we are using "primary" in this tutorial, for example). In other documentation and discussions, you may see "master", or other terms, used to refer to the primary branch. Regardless of the name, just keep in mind that nearly every repository has a primary branch that can be thought of as the official version of the repository. If it's a website, then the primary branch is the version that users see. If it's an application, then the primary branch is the version that users download. This isn't *technically* necessary (git doesn't treat any branches differently from other branches), but it's how git is traditionally used in a project.

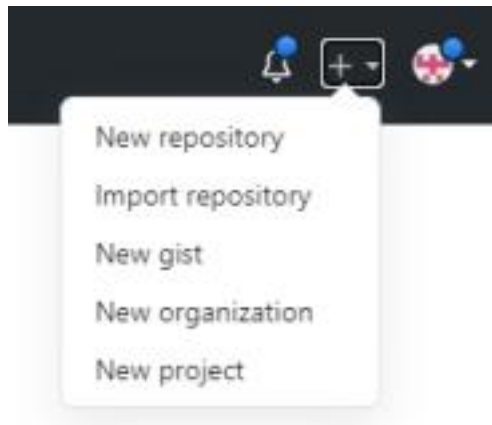
If you are curious about the decision to use different default branch names, GitHub has an explanation of *their* change here: <https://github.com/github/renaming>

Now, if you switch back to the primary branch and make some more commits, your new branch won't see any of those changes until you [merge](#) those changes onto your new branch.

Step 6: Create a new repository on GitHub

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

To create a new repo on GitHub, log in and go to the GitHub home page. You can find the “New repository” option under the “+” sign next to your profile picture, in the top right corner of the navbar:



After clicking the button, GitHub will ask you to name your repo and provide a brief description:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

DoQuocBinh ▾

/

Repository name *

MyCoolProject ✓

Great repository names are short, lowercase, and unique. [MyCoolProject is available.](#) d inspiration? How about [stunning-octo-barnacle?](#)

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:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

When you're done filling out the information, press the 'Create repository' button to make your new repo.

GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the '...or push an existing repository from the command line' section:

Run the commands which Github gives you

...or push an existing repository from the command line

```
git remote add origin https://github.com/DoQuocBinh/MyCoolProject.git
git branch -M main
git push -u origin main
```



```

C:\Users\Do Quoc Binh\MyFirstRepo>git remote add origin https://github.com/DoQuocBinh/MyCoolProject.git

C:\Users\Do Quoc Binh\MyFirstRepo>git branch -M main

C:\Users\Do Quoc Binh\MyFirstRepo>git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 102.91 KiB | 14.70 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/DoQuocBinh/MyCoolProject.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

```

(You'll want to change the URL in the first command line to what GitHub lists in this section since your GitHub username and repo name are different.)

Step 7: Push a branch to GitHub

Create a new Branch

Switch to the new Branch

Add a file to a new Branch(Security.pdf)

View the status

```

C:\Users\Do Quoc Binh\MyFirstRepo>git branch my-new-branch

C:\Users\Do Quoc Binh\MyFirstRepo>git checkout my-new-branch
Switched to branch 'my-new-branch'

C:\Users\Do Quoc Binh\MyFirstRepo>git status
On branch my-new-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Security.pdf

nothing added to commit but untracked files present (use "git add" to track)

```

Add changes

Commit changes(Security.pdf) to the new branch

Push changes to remote


```

C:\Users\Do Quoc Binh\MyFirstRepo>git add .

C:\Users\Do Quoc Binh\MyFirstRepo>git commit -m 'just copy security.pdf'
error: pathspec 'copy' did not match any file(s) known to git
error: pathspec 'security.pdf' did not match any file(s) known to git

C:\Users\Do Quoc Binh\MyFirstRepo>git commit -m "just copy security.pdf"
[my-new-branch d81c1ec] just copy security.pdf
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Security.pdf

C:\Users\Do Quoc Binh\MyFirstRepo>git push origin my-new-branch
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 817.60 KiB | 22.71 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'my-new-branch' on GitHub by visiting:
remote:   https://github.com/DoQuocBinh/MyCoolProject/pull/new/my-new-branch
remote:
To https://github.com/DoQuocBinh/MyCoolProject.git
 * [new branch]      my-new-branch -> my-new-branch

```

Now there will be to branches with different content

The screenshot displays the GitHub web interface for a repository named 'MyCoolProject' by user 'DoQuocBinh'. It shows two branches side-by-side for comparison.

Top Branch: my-new-branch

- Status: 1 commit ahead of main.
- Commit: d81c1ec, 8 minutes ago, 2 commits.
- Files:
 - Mongoose 101.docx: Your message about the commit (1 hour ago)
 - Security.pdf: just copy security.pdf (8 minutes ago)
- Footer: "Help people interested in this repository understand your project by adding a README." with a green "Add a README" button.

Bottom Branch: main

- Status: 1 commit behind my-new-branch.
- Commit: f2f1f5, 1 hour ago, 1 commit.
- Files:
 - Mongoose 101.docx: Your message about the commit (1 hour ago)


Step 8: Create a pull request (PR)

A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the primary branch.

This is what the PR page looks like before you've submitted it:

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](#).



base: main

←

compare: my-new-branch

✓ Able to merge. These branches can be automatically merged.

just copy security.pdf

Write

Preview

H

B

I

≡

<>

🔗

≡

≡

📄

👤

📧

↶

I want to add this to main

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

Create pull request

And this is what it looks like once you've submitted the PR request:

You might see a big green button at the bottom that says 'Merge pull request'. Clicking this means you'll merge your changes into the primary branch..

Sometimes you'll be a co-owner or the sole owner of a repo, in which case you may not need to create a PR to merge your changes. However, it's still a good idea to make one so you can keep a more complete history of your updates and to make sure you always create a new branch when making changes.

Step 9: Merge a PR

Go ahead and click the green 'Merge pull request' button. This will merge your changes into the primary branch.

When you're done, I recommend deleting your branch (too many branches can become messy), so hit that grey 'Delete branch' button as well.

You can double check that your commits were merged by clicking on the 'Commits' link on the first page of your new repo.

This will show you a list of all the commits in that branch. You can see the one I just merged right up top (Merge pull request #1).

You can also see the [hash code](#) of the commit on the right hand side. A hash code is a unique identifier for that specific commit. It's useful for referring to specific commits and when undoing changes (use the [git revert](#) <hash code number> command).

command to backtrack).

Step 10: Get changes on GitHub back to your computer

Right now, the repo on GitHub looks a little different than what you have on your local machine. For example, the commit you made in your branch and merged into the primary branch doesn't exist in the primary branch on your local machine.

In order to get the most recent changes that you or others have merged on GitHub, use the `git pull origin master` command (when working on the primary branch). In most cases, this can be shortened to “`git pull`”.

This shows you all the files that have changed and how they've changed. Now we can use the `git log` command again to see all new commits.

(You may need to switch branches back to the primary branch. You can do that using the `git checkout master` command.)

Step 11: Bask in your git glory

You've successfully made a PR and merged your code to the primary branch. Congratulations! If you'd like to dive deeper, check out these more advanced tutorials and resources:

Introduction to GitHub desktop

[Creating your first repository using GitHub Desktop - GitHub Docs](#)