

# An Introduction to Git and GitHub

Zheng Tian

## 1 What are Git and GitHub?

### 1.1 What is Git?

- Git is a version control system (VCS) that records changes to a file or set of files over time so that you can recall specific versions later.
- Using Git to manage versions means that if you screw things up or lose files, you can easily recover.

### 1.2 How does Git work?

- Git takes a snapshot of a whole project whenever you make a commit to the changes.

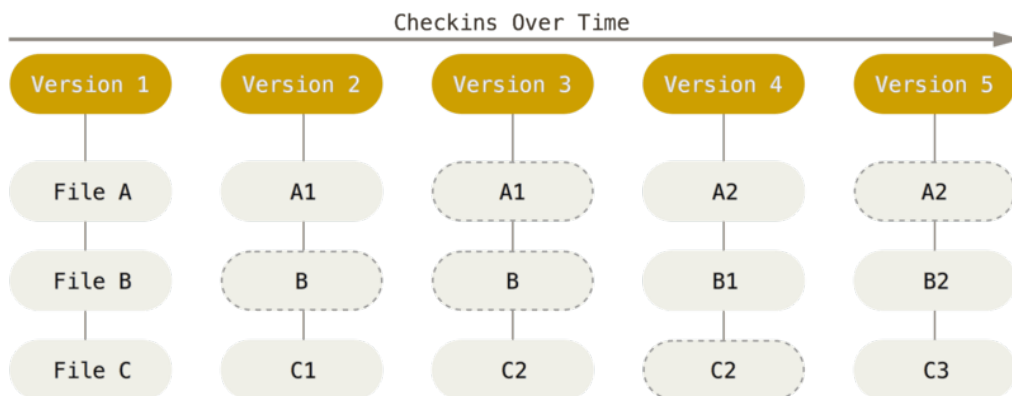


Figure 1: Storing data as snapshots of the project over time

### 1.3 The basic workflow of Git goes like this

1. You modify files in your working tree.
2. You stage the files, adding snapshots of them to your staging area.

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

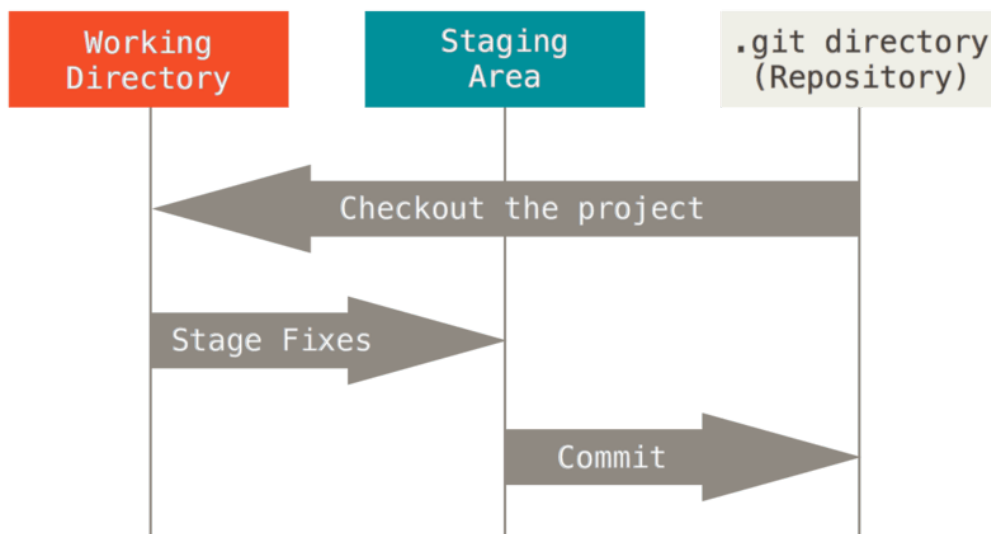


Figure 2: Working tree, staging area, and Git directory

## 1.4 What is GitHub?

- GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects.
- Visit <https://github.com/> and sign up a free account right now!

## 2 Get Started

### 2.1 Install Git and GitHub Desktop

#### Download and install Git

You can download the installers of Git for Windows, MacOS, and Linux at <https://git-scm.com/downloads>.

#### (Optional) Download and install GitHub Desktop

- GitHub Desktop is a GUI for Git and integrate your repositories on GitHub with local counterparts.
- Download the installer of GitHub Desktop at <https://desktop.github.com/>.

## 2.2 Configure Git

Next, follow the steps of configuration at <https://help.github.com/articles/set-up-git/>.

**Open a terminal.**

- In MacOS, search `terminal` in Spotlight Search.
- In Windows, open `Git Shell` that is installed with GitHub Desktop. (Or `Git Bash` if you do not install GitHub)

**Configure user name and user email**

It is wise to use the same user name and email when you sign up with GitHub.

```
$ git config --global user.name "YOUR NAME"
$ git config --global user.email "EMAIL ADDRESS"
```

**(Optional) Setup a default editor**

GitHub recommend using Atom.

- Download Atom at <https://atom.io/>.
- Configure Atom to be the default editor of Git, as instructed at <https://help.github.com/articles/associating-text-editors-with-git/>

In MacOS,

```
$ git config --global core.editor "atom --wait"
```

In Windows

```
$ git config --global core.editor
"C:/Users/USERNAME/AppData/Local/atom/app-VERSION/atom.exe"
```

## 2.3 Book: *Pro Git*

Git is fully documented in this book, <https://git-scm.com/book/en/v2>.

## 3 Git Basics

Let's introduce basic command of Git through setting up a demo project.

### 3.1 Create a Git repository

A Git repository is simply a folder containing the files pertaining to a project that you want to have versions controlled. Within the folder, there is a sub-folder, usually invisible, called `.git`, where all data regarding each different version are stored.

#### Create a Git repository in GitHub

- At <https://github.com>, click **New Repository** and follow the instruction to create a new repository, names `demo_project`. You may check the box saying "Initialize this repository with a README". This repository is now at GitHub not in your computer.
- To make it in your computer, the quickest way is to press the button of **Set up in Desktop**, which will call GitHub Desktop and set up the repository in your computer.

Another way is to copy the **HTTPS** of the repository and type the following command, assuming that you are under a folder in which you want this new repository to be downloaded.

```
$ git clone https://github.com/YOUR_USERNAME/demo_project.git
$ cd demo_project
$ ls -a
```

- As shown in Figure 3, there are other methods to create a new repository.

#### Create a Git repository in a computer

We can also start creating a Git repository in our computer and then push it to GitHub.

- Create a new folder, called `tmp_project`
- Go to this folder
- Make it a git repository.

```
$ mkdir git_test
$ cd tmp_project
```

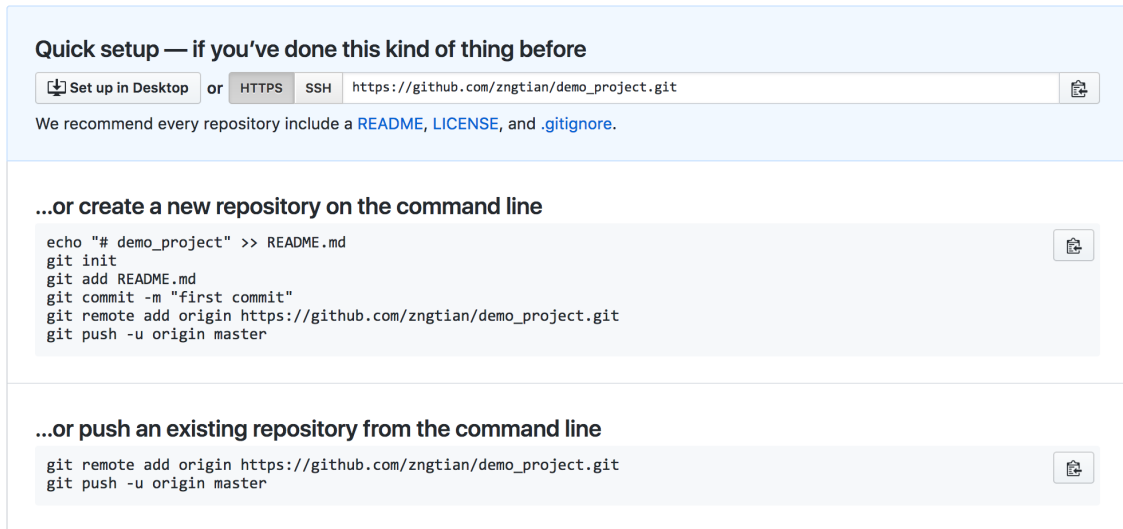


Figure 3: Create a Repository in GitHub

```

$ git init
$ git status
  
```

- We use `git status` to check the status of the repository.

To push this repository to GitHub, first we create a new repository at GitHub, called `tmp_project`.

```

git remote add origin https://github.com/zngtian/tmp_project.git
git push -u origin master
  
```

### 3.2 A life-cycle of a file in a git repository

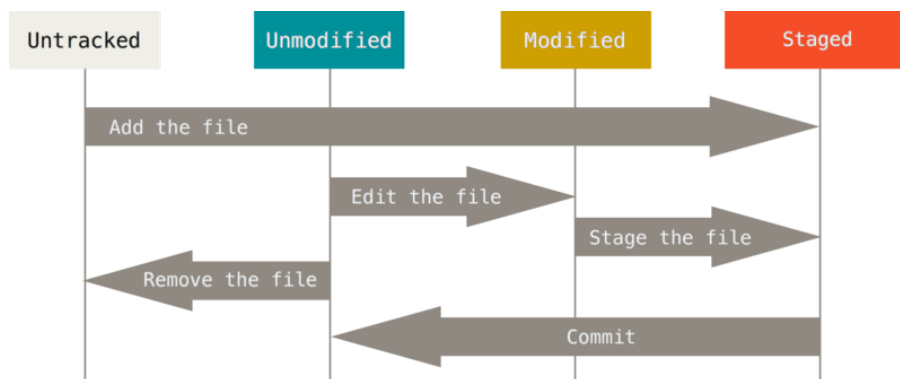


Figure 4: The lifecycle of the status of your files

Figure 4 displays a life cycle of a file in a git repository.

- When a file is first created, its status in git is **untracked**. For example, we create a new file `testing_file_1.txt`.
- We want git to know that we have added a new file. With git, we need to **stage** the file.
- After we have made all changes regarding the new file, we need to tell git that all changes are done. With git, we need to **commit** the file.
- After committing, if we make any change with `testing_file_1.txt`, the status of this file changes to **unstaged**, we need to repeat staging and committing to make the change accepted by Git.

### 3.3 Add a new file, stage it, and commit it

Now add a new file in the local `demo_project` repository in our computer, called `firstfile.txt`. After adding the new file, type the following command.

```
$ git status
```

Now the new file is in working directory, but it is not staged. To stage it, type

```
$ git add firstfile.txt
```

```
$ git status
```

Now the file is staged and ready to be committed. Type the following command,

```
$ git commit -m "add a first file"
```

```
$ git status
```

We have two files in the repository, `README.md` and `firstfile.txt`. Let's make other changes in these two files, and check the status. To stage all these changes, use `git add -A`.

```
$ git status
```

```
$ git add -A
```

### 3.4 View history and difference

When a file is changed but not staged, we can use

```
$ git diff
```

When it is staged, we can use

```
$ git diff --staged
```

Sometimes, we need to review the whole history of what we have done. Use the following commands

```
$ git log
$ git log -p -2
$ git show [commit]
```

### 3.5 Undo changes

Git as a version control tool can easily help us change a modified file back to its past status.

Let's change `firstfile.txt`. Now the change is not staged. If you do not want such a change, you can type

```
$ git checkout -- firstfile.txt
```

Let's continue to make a change and stage it. If you regret making such a change again, you can type

```
$ git reset HEAD firstfile.txt
$ git checkout -- firstfile.txt
```

If you have commit a change and you want the file to change back to a previous commit, you can type

```
$ git reset --hard <commit>
```

### 3.6 Upload your work to GitHub

The local repository has moved to a new commit, but its counterpart at GitHub is still staying at the original status. We can upload the new file to GitHub by typing the following

```
$ git push
```

### 3.7 Create a new branch

Now the demo project has only one branch, `master`. Let's create a new branch, called `testing`.

```
$ git branch testing
$ git checkout testing
```

In this new branch, we add a new file, named `testing_file.txt`, stage it, and commit.

```
$ git add testing_file.txt
$ git commit -m "add a testing file"
$ git status
$ git branch -v
```

This new file is added in the `testing` branch. But it is not in the `master` branch.

```
$ git checkout master
$ git merge testing
```

Creating a branch gives us a way to test something and if it is satisfactory, we can add it to the main branch that is `master` in this case.

Now the local `master` branch has been updated. We can update the Github repository.

```
$ git status
$ git push
```

## 4 Group Working on a Project

Up to now, we are working on the demo project on our own. We can now track the changes we did with the project, create branches to test new things, and upload our work to GitHub. However, GitHub is more than a cloud-like storage for our own work. It is a platform for facilitating cooperation in teamwork.

Let's continue with the demo project. A new member wants to collaborate in the project. The creator of the project is the maintainer of the project.

### 4.1 Add a collaborator

For the maintainer of the project, on the page of `demo` project on GitHub, click **Settings** and **Collaborators**, as shown in Figure 5. Then, search a collaborator by his GitHub user name and add him. An email will be sent to the collaborator, waiting for his acceptance. Upon accepting the invitation, the collaborator will have the push and pull right to the project.



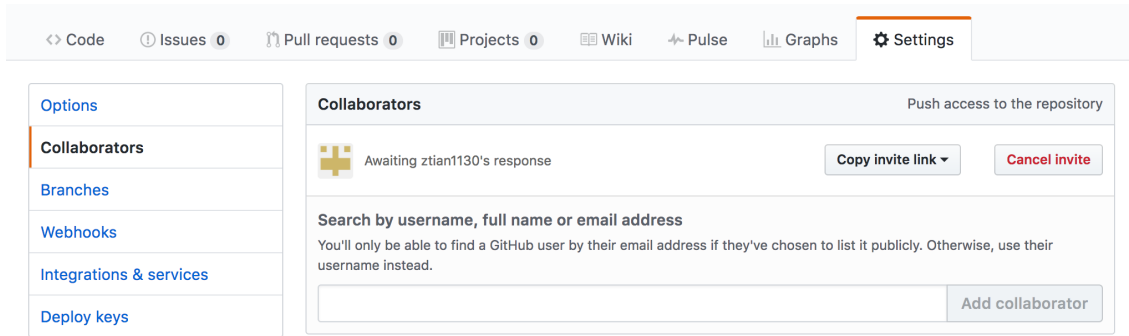


Figure 5: Add a collaborator

## 4.2 Fork and clone the project

Even though the collaborator can clone the `demo_project` repository directly, the GitHub-favored workflow is forking and cloning. Forking a repository is to make that repository be your own copy so that you can work on that and to some extent avoid unintended damages to the original repository.

Upon accepting the invitation, the web browser will direct the collaborator to the maintainer's repository. What he can do is looking around and forking by clicking the **Fork** button.

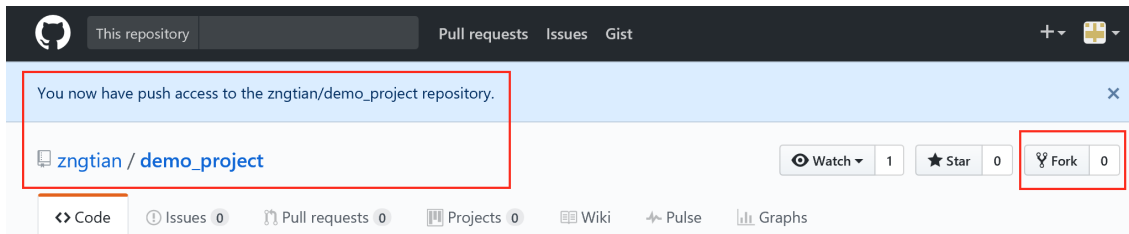


Figure 6: Fork a project

Then on the collaborator's own GitHub account, he has a forked repository from which he can then clone.

```
$ git clone https://github.com/ztian1130/demo_project.git
```

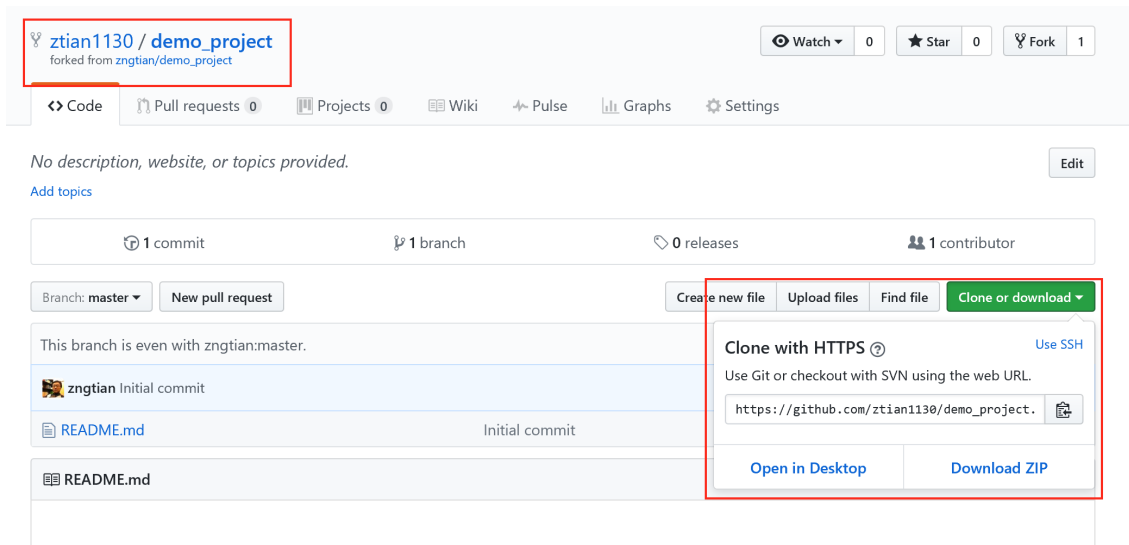


Figure 7: The forked project

### 4.3 Collaborator making changes

Now the collaborator wants to make some changes. What he needs to do is to create a new branch, make some changes, and push it to his forked repository. For example, in README he adds his name.


```
$ git branch zt1130
$ git checkout zt1130
$ git commit -a -m "add my name in README"
$ git push origin zt1130
```

### 4.4 Pull requests

The concept of pull requests is the key of collaboration in GitHub. After the collaborator pushes the new branch to GitHub, the webpage of the forked repository will show a block pointing you to **compare and pull request**. Then, click the green button and open a pull request. The following page, shown in Figure 8, will then show up.

## 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 fork: **zngtian/demo\_project** ▼

base: **master** ▼

...

head fork: **ztian1130/demo\_project** ▼

compare: **zt1130** ▼

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



add my name in readme

Write

Preview

AA ▼ B i “ < > 🔗 ⋮ ≡ ≡ ≡ ↶ @ 🚩

I wrote a new line in README.md, and add our names. |

Attach files by dragging & dropping or [selecting them](#).

☒ **Allow edits from maintainers.** [Learn more](#)

Create pull request

Figure 8: Open a pull request

GitHub automatically checks whether there are any conflicts. And you can type in more information to tell the maintainer what you did. Then, click **Open pull request**.

GitHub will send the maintainer an email notifying him that a collaborator has made a pull request. In fact, since both the maintainer and the collaborator have the push and pull right, the collaborator himself can decide to merge the pull request. But now let the maintainer do the job.

The maintainer opens the pull request and decides to merge since there is no conflict, as shown in Figure 9. The pull request is closed after merging is complete. And the maintainer's master branch is up to date. (But the collaborator's master branch is not updated. See the next section)

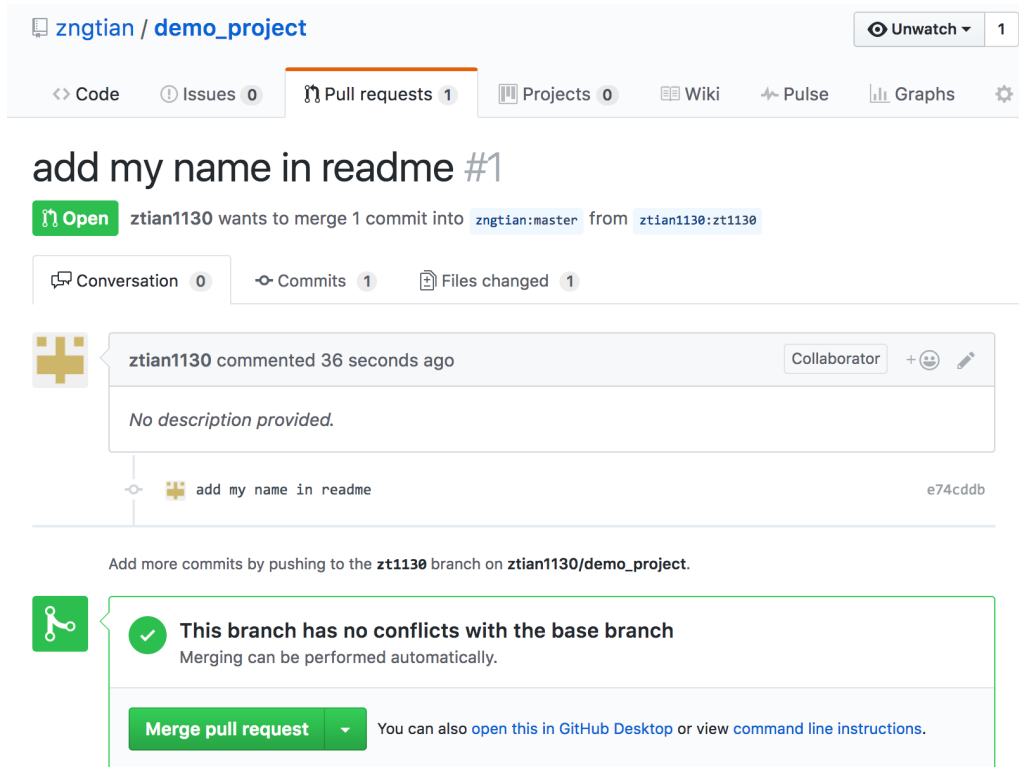


Figure 9: Merge pull request

## 4.5 Keep up with the main repository

Since there is a change in the master branch of the maintainer's GitHub repository, both the maintainer and the collaborator need to pull the latest work down.

- For the maintainer, do this

```
$ git fetch
$ git merge origin/master
```

- For the collaborator, do this

```
$ git checkout master
$ git remote add main https://github.com/zngtian/demo_project.git
$ git fetch main
$ git merge main/master
```

Now both the maintainer and the collaborator have the latest version of the project. They can move on to more work.

## 4.6 The basic workflow of teamwork

1. Before making new changes, make sure you have the latest development by using `git fetch` or `git pull`.
2. Create and checkout to a topic branch to make changes, and commit.
3. Upload the topic branch to GitHub and open a pull request.
4. Check whether the changes lead to conflicts or whether more changes need to be done before merging.
5. Merge the changes to the main repository when the maintainer and the collaborator both agree with the changes.
6. Update the local repository using `git pull` at the local `master` branch.
7. Repeat 1-6 for future work.