Git and GitHub for Collaboration: a step-by-step tutorial

leifengblog.net/blog/git-and-github-for-collaboration-a-step-by-step-tutorial

06 Sep 2023 | Tutorial | Git • GitHub

This post is a quick start manual for beginners to use Git with GitHub for source code control. Although this manual aims for the operations with repositories of GitHub, the instructions apply to general Git practices. This post assumes that you have already set up a GitHub account where you can store your source code and manage the development.

Table of Content

Introduce to Git and GitHub

Git is an open source and free tool for source control management or what's referred to as SCM. With Git, you can manage changes to files over time, so you can easily revert to a previous version if necessary. Git is also a distributed version control system, which means that you can work on your code on your own computer and then easily share it with others.

GitHub is a hosting service for Git repositories. It provides a place to store your code, collaborate with others, and track issues. GitHub is also a social platform where you can connect with other developers and learn from them.

Git and GitHub are essential tools for software development. They help us to track changes to our code, collaborate with others, and learn from others.

Install and Configure Git

Many operating systems come with Git pre-installed, especially Linux and macOS systems. To check if Git is already installed on your system, open a terminal window and type the following command:

```
git --version
```

If you need to install a Git for your operating systems, please follow instructions from Installing Git.

Here are the major steps of installation for Windows and CentOS Linux:

Download and Install Git for Windows

1. Go to https://git-scm.com/download/win download the lastest version. The download will start automatically. If not, you can manually click the corresponding download links.

1. When the download finishes, you will get an `.exe` file, and you can double-click it to install. You can keep all default settings during installation.

Install Git on CentOS Linux

The easiest way to install Git on CentOS Linux is through CentOS's package manager - Yum(Yellowdog Updater, Modified):

```
sudo yum install git
```

Configure Git for the First Time

After installation of Git on your system, if you are using Windows system, you can start `Git Bash` from the **Start Menu**:

You can also start `Git Bash` from the **Context Menu** through right-clicking from any local directory while pressing the **shift** button:

You can also use the default command line of your operating system to run git commands. For the first time to run git command, it is a good idea to configure your Git environment by running the following commands:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

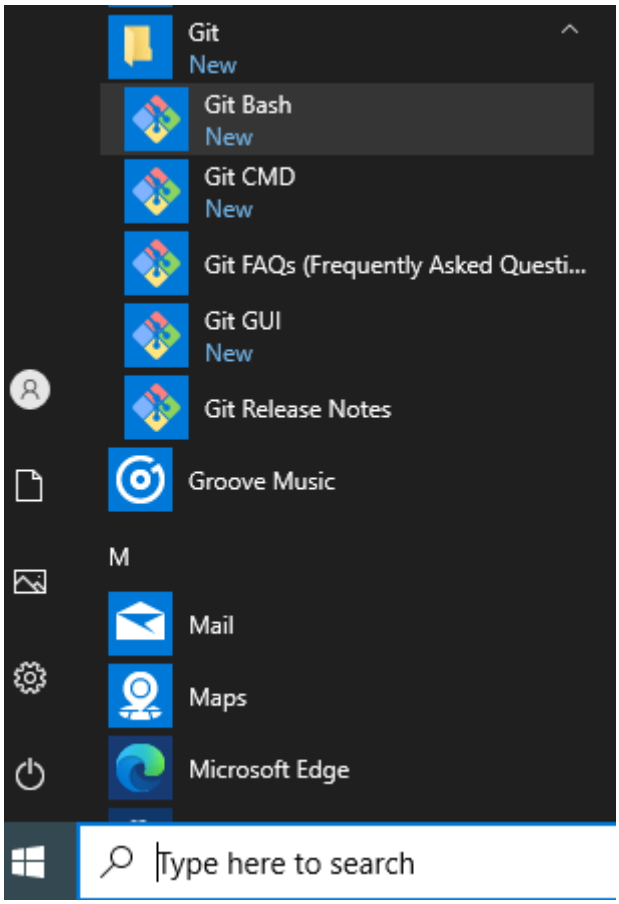
If you need set up proxy for git, you can run the following commands:

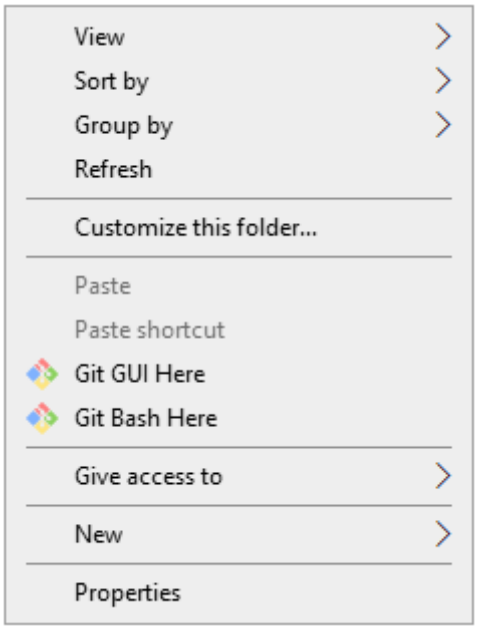
```
git config --global http.proxy <proxy_url>
git config --global https.proxy <proxy_url>
```

You can check all your settings through running:

```
 git config --list
```

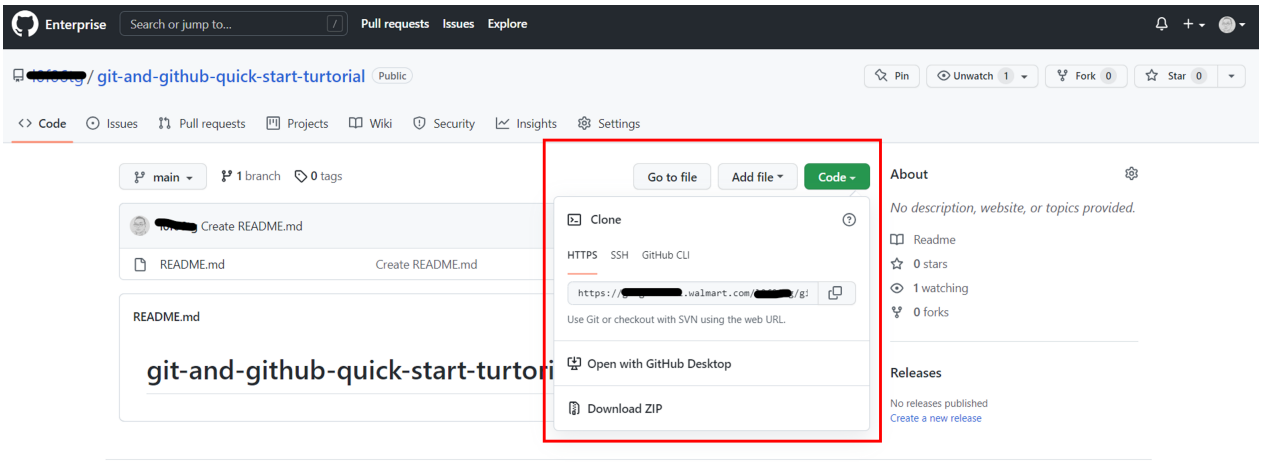
This command looks for the file of `.gitconfig` from your $HOME directory. On Windows, the $HOME direcotry is usually `C:\Users\$USER`. On Linux, you can check your $HOME directory through command `echo $HOME`.





Clone a Code Repository from GitHub

You can get a copy of a remote repository from GitHub through the `git clone` command. Click on the **Code** button shown in the following screenshot. There are two ways to connect to the remote repository: **HTTPS** and **SSH**.



Clone with HTTPS

1. Go to the GitHub repository that you want to clone.
2. Click on the **Code** button.
3. Under the HTTPS tab, copy the URL.
4. In your local terminal, navigate to the directory where you want to clone the repository.
5. Run the following command:

```
git clone <URL_HTTPS>
```

The command would prompt a message asking for the username and password to the remote GitHub repository. Enter your credentials and press Enter.

The remote repository will be cloned to your local directory.

Note: Cloning a repository using HTTPS is not the recommended way to do it. This is because it requires you to enter your GitHub username and password every time you want to clone the repository. This can be a security risk, especially if you are cloning the repository from a public computer.

Clone with SSH

A more secure and convenient way to clone a repository is to use SSH. SSH is a secure protocol that allows you to authenticate to a remote server without having to enter your username and password. To use SSH, you will need to generate an SSH key pair. Once you have generated your SSH key pair, you can add it to your GitHub account. Then, you can clone the repository using the following command:

```
git clone git@<GITHUB_SERVER>.com:<USERNAME>/<REPOSITORY_NAME>.git
```

Generate a SSH Key Pair

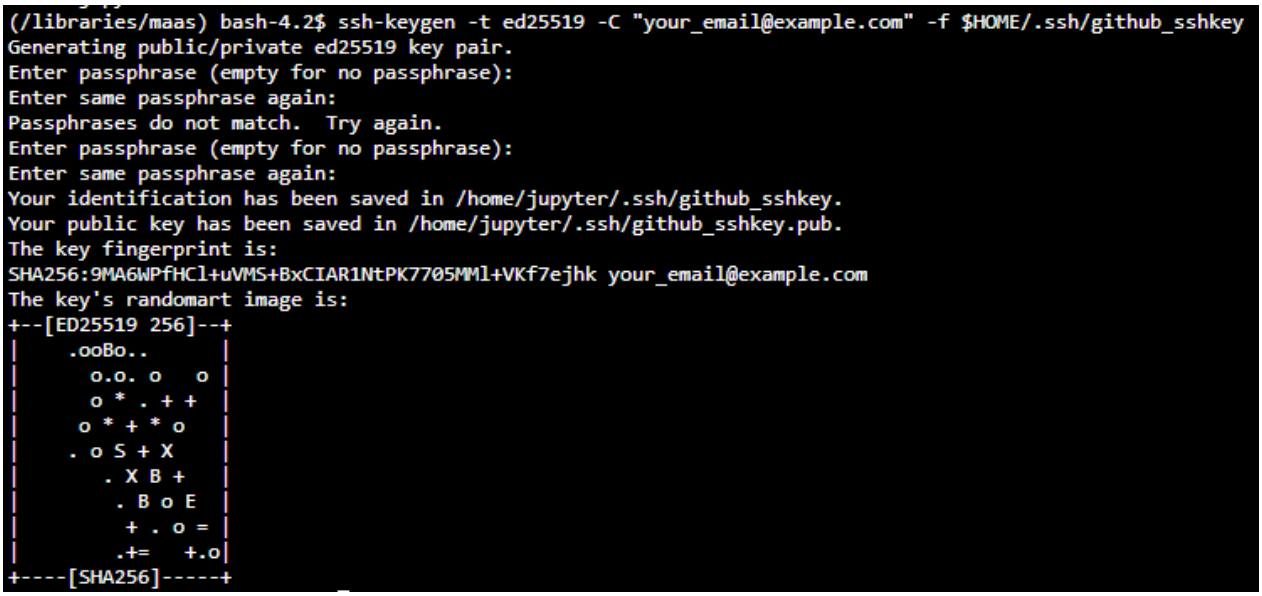
You can create a new SSH key pair by running the following command in your local terminal.

```
ssh-keygen -t ed25519 -C "your_email@example.com" -f $HOME/.ssh/<your file name>
```

You can just press Enter to use empty passphrase when the command prompt to ask for passphrase. This command generates two files with names of `<your file name>` and `<your file name>.pub` under the path of `$HOME\.ssh\`.

Note: if you are using a legacy system that doesn't support the Ed25519 algorithm, use:.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com" -f $HOME/.ssh/<your file name>
```



Adding SSH Key to SSH Agent

Before adding a new SSH key to the ssh-agent to manage your keys, you should have checked for existing SSH keys and generated a new SSH key.

Run the following command to start the ssh-agent in the background:

```
eval `ssh-agent -s`
```

Run the following command to add your SSH private key to the ssh-agent.

```
ssh-add ~/.ssh/<your file name>
```

Add the following settings into ~/.ssh/config file:

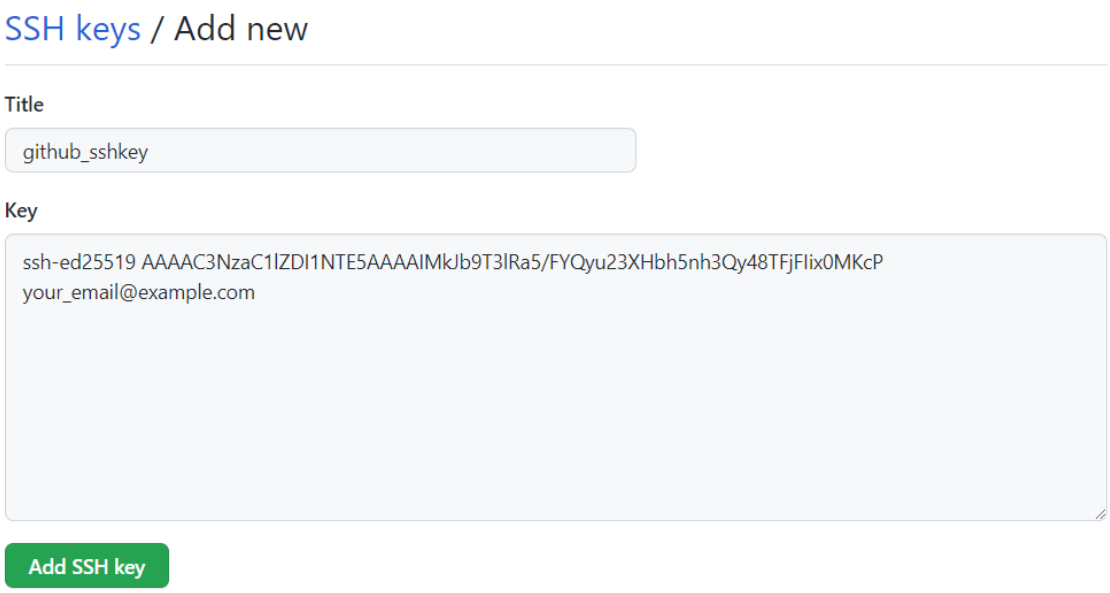
```
# my company gitlab production environment
Host *
  PreferredAuthentications publickey
  AddKeysToAgent yes
  IdentityFile ~/.ssh/ado_rsa
```

Add the Public Key GitHub Settings

Once you get the SSH key pair, add the SSH public key to your GitHub account settings:

1. Log into GitHub, click your profile photo icon in the upper-right corner of any page, then select **Settings** from the dropdown list.
2. In the "Access" section of the left sidebarSelect, click **SSH and GPG keys**

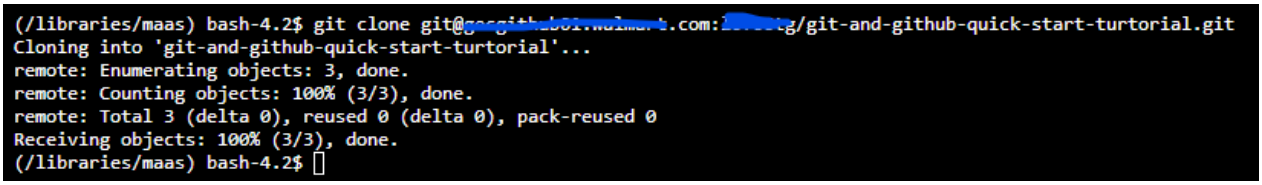
3. Click **New SSH key** button in the upper-right corner of the page
4. Copy the content from generated public key file `<your file name>.pub` and paste it in the **Key** box and give the key an informative title for future management
5. Click **Add SSH key**.



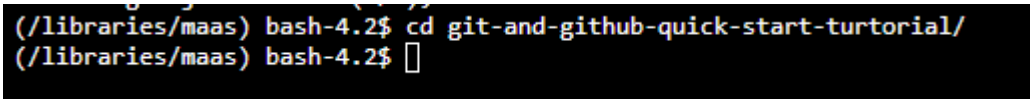
Clone with SSH Command

Once you set up your SSH key in your GitLab profile, you can run the following command with the corresponding SSH URL to clone the remote repository without inputting your username and password.

```
git clone <URL_SSH>
```

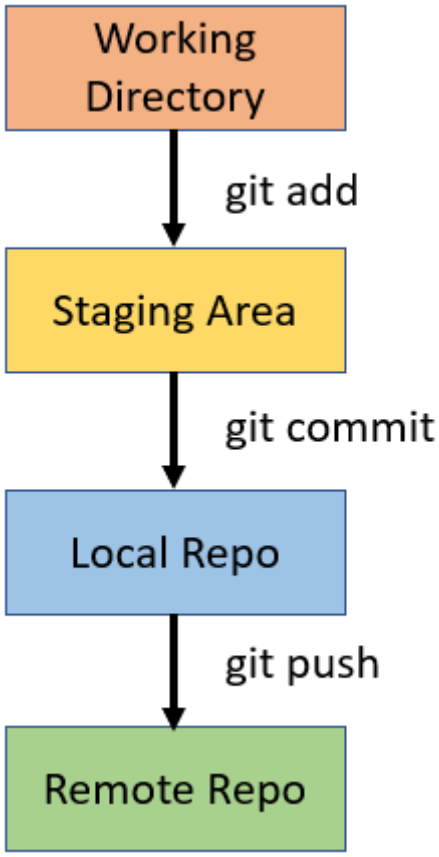


Once you clone the remote repository to local successfully, you can navigate to the directory through `cd <REPOSITORY_NAME>`:



Now, you are inside your working directory, and you can make any changes you like. Once you are happy with the changes, you can `add` and `commit` them to the local repository and then `push` to the remote repository so that your fellow team members can see the changes you made.

Please note that you are in the default **main** branch and all the changes you made are within this branch. And when you push the changes to the remote repo, it will update the **main** branch there. Usually, this is not the best practice.



Best Practices for Code Review

Excellent code depends on rigorous review. The following flow are usually the best practices for code review and team collaboration:

- Once checks out a repository, a developer creates a new feature **branch**, makes changes in this feature branch, and tests it.
- When happy on the changes, the developer **push**es the changes in the **new branch**, and make a **pull request**.
- The developer assigns the **pull request** to a **reviewer**, who looks at it and makes comments as needed. When the reviewer finishes, he/she can assign it back to the author.
- The author **addresses the comments**. This stage can go around for a while, but once both reviewer and author are happy, they can **approve a merge** or assign it to a final reviewer who can do the merge.
- The final reviewer follows the same reviewing process again. The author again addresses any comments. Once the final reviewer is happy and the build is green, then the new branch will be **merged** with the target branch.

Here are the demos on each of the code review steps.

Pull from Remote

Before you make any changes, it is always a good idea to run the command `git pull` within the local repository to make sure your current local version is up to date.

```
(/libraries/maas) bash-4.2$ pwd
/home/jupyter/git-and-github-quick-start-turtorial
(/libraries/maas) bash-4.2$ git pull
Already up to date.
(/libraries/maas) bash-4.2$
```

Create a New Branch

Branches let you work on new features or bug fixes of the main project code in the `master` branch. You can use `git branch <BRANCH_NAME>` to create a new branch and `git branch` to check your branches.

```
(/libraries/maas) bash-4.2$ git branch lei_test
(/libraries/maas) bash-4.2$ git branch
  lei_test
* main
(/libraries/maas) bash-4.2$
```

Note the asterisk next to the `main`, which indicates that you are currently in the `main` branch. To switch to your new branch, you need to run `git checkout <BRANCH_NAME>`.

```
(/libraries/maas) bash-4.2$ git checkout lei_test
Switched to branch 'lei_test'
(/libraries/maas) bash-4.2$ git branch
* lei_test
  main
(/libraries/maas) bash-4.2$
```

Commit Changes

Now, you are in the new feature branch and ready to change your codes.

git status

Once you finish your changes, you can run `git status` to check the changes you have made.

```
(/libraries/maas) bash-4.2$ git status
On branch lei_test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

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

no changes added to commit (use "git add" and/or "git commit -a")
(/libraries/maas) bash-4.2$
```

git add

You can run the command `git add <FILE_NAME>` (add the specific file) or `git add .` (add all files) to add your changes to the staging area. A staging area is an intermediate place between your working directory and local Git repo where any changes that you've made can be reviewed before you actually commit them to the repo.

```
(/libraries/maas) bash-4.2$ git add .
(/libraries/maas) bash-4.2$ git status
On branch lei_test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .ipynb_checkpoints/README-checkpoint.md
        modified:   README.md

(/libraries/maas) bash-4.2$
```

git commit

Next, you can commit the changes to your local repo through running `git commit -m "MESSAGE_TEXT"`.

```
(/libraries/maas) bash-4.2$ git commit -m "updated the turtorials"
[lei_test 01064e3] updated the turtorials
 2 files changed, 3 insertions(+)
 create mode 100644 .ipynb_checkpoints/README-checkpoint.md
(/libraries/maas) bash-4.2$
```

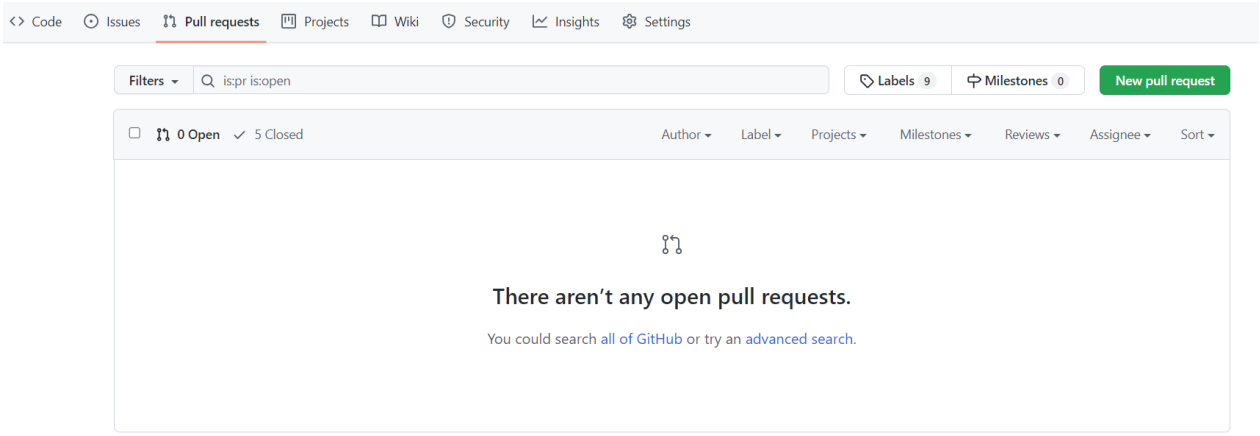
git push

After committing to the local repo, you can push the commits from the local to the new branch in the remote repo through running `git push -u origin <BRANCH_NAME>`.

```
(/libraries/maas) bash-4.2$ git push -u origin lei_test
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 72 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 441 bytes | 31.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'lei_test' on GitHub by visiting:
remote:      https://gecgithub01.walmart.com/l0f06tg/git-and-github-quick-start-turtorial/pull/new/lei_test
remote:
To gecgithub01.walmart.com:l0f06tg/git-and-github-quick-start-turtorial.git
 * [new branch]      lei_test -> lei_test
branch 'lei_test' set up to track 'origin/lei_test'.
(/libraries/maas) bash-4.2$
```

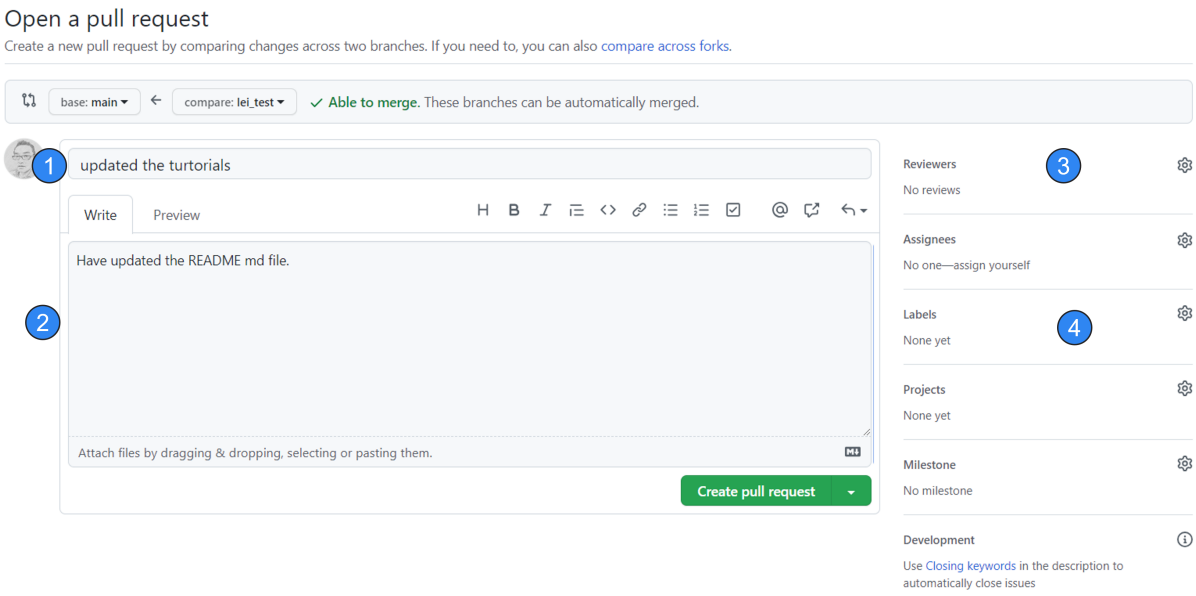
merge request

Once you push the local to the new branch in the remote repository, you can submit a pull request through **Pull Requests** tab and then **New pull request** button on the remote repository page.



The **New pull request** button will pop up the **Open a pull request** page, where you can:

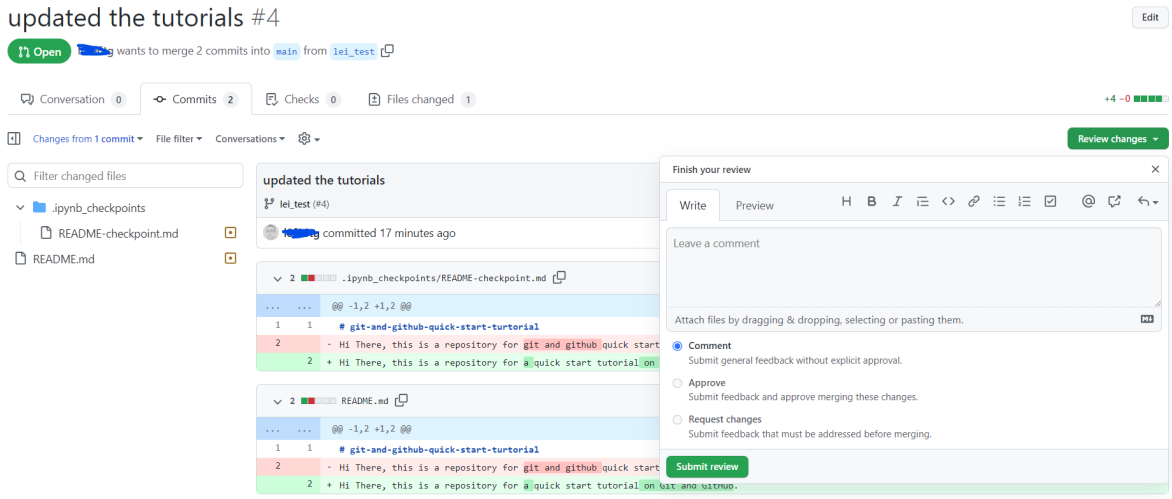
1. fill the title of the request;
2. write a brief description or comments;
3. assign reviewer(s) from the drop down list;
4. add tags and labels;
5. link work item(s) from the drop down list;



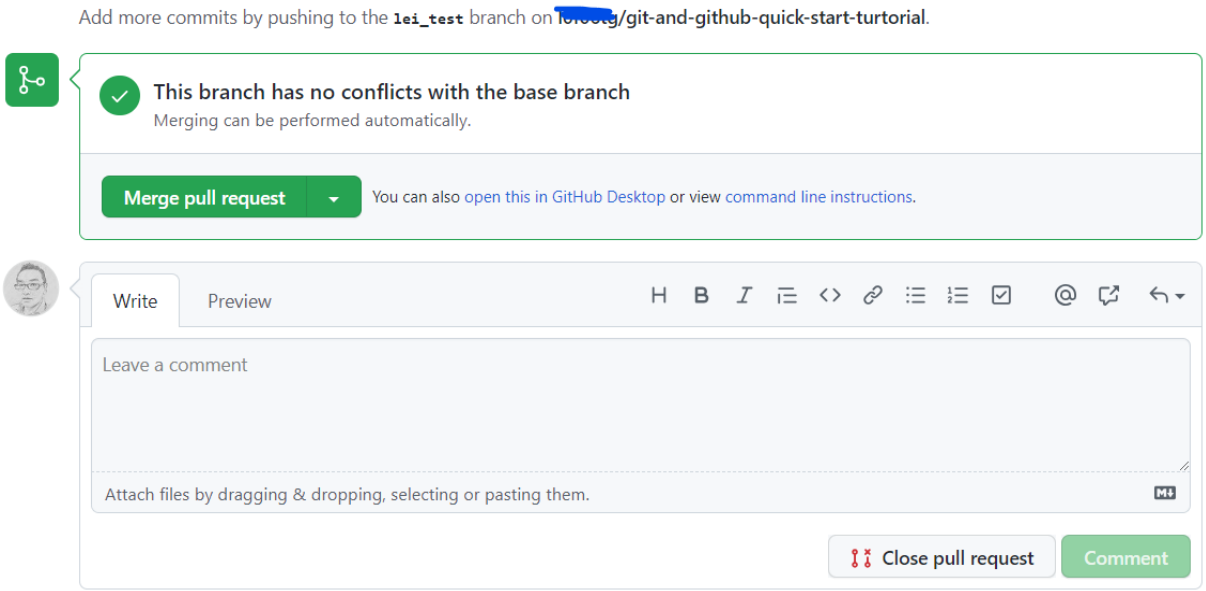
Once the pull request has been submitted, the reviewer(s) will get notification and they can go the request page to do:

1. check the commits and changes;
2. write comments;
3. approve the merge (if applicable);

4. request changes (if applicable).

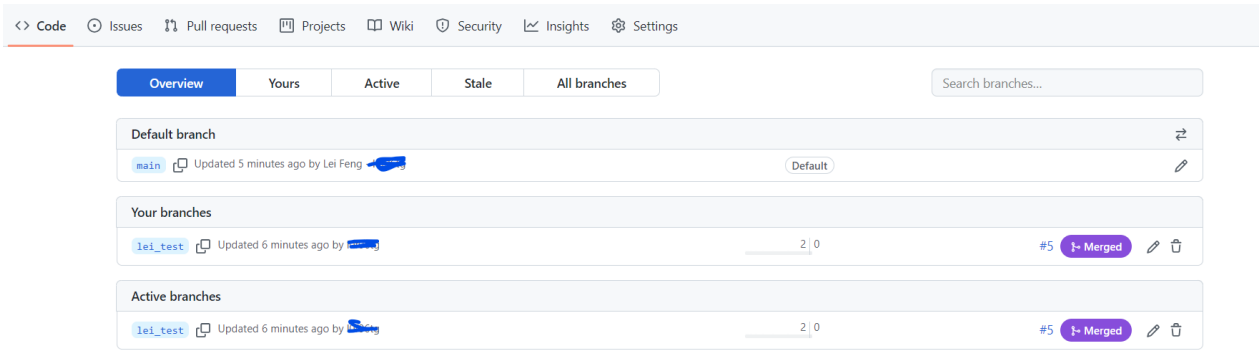


After the changes have been approved, the branch can be merged to the target branch through the **Merge pull request** button.

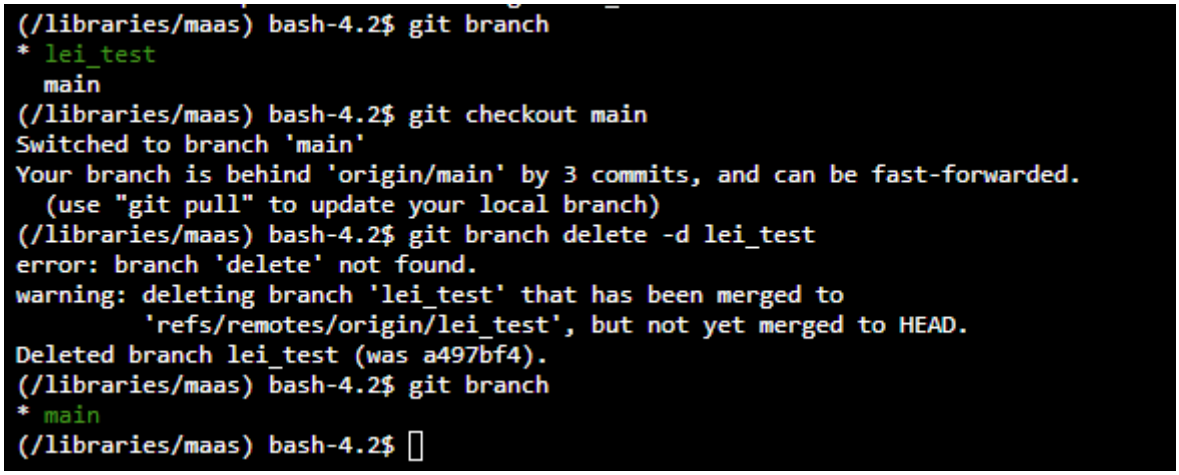


delete the merged branch

Once your new branch has been merged to the target branch in the remote repository, you can delete the merged branches from the **branches** tab in the GitHub repository.



You can delete branch from your local repository by running `git branch -D <BRANCH_NAME>`. Note, you have to get out of the branch before you run this command deleting it.



pull from remote

Now, it will be a good practice to run `git pull` again to synchronize your remote repository to your local.

I hope this quick-start tutorial is helpful to you. Please feel free to leave any comments and advice. Thanks for reading.

References