Git Commands to Work on GitHub

Contents

As research becomes increasingly collaborative and multiple people work on the same project, it becomes difficult to keep track of changes made by others if not done systematically. Moreover, it is time-consuming to manually incorporate the work of different participants in a project, even when all of their changes are compatible. Hosting the project on an online repository hosting service like GitHub is beneficial to make collaborations open and effective. If you are new to collaboration through GitHub, please follow the comprehensive guide in the previous sections.

In this section, we will discuss how to use Git commands to work with an online Git repository.

Please note that the commands listed in this chapter (both in this and previous subchapters) are NOT specific to GitHub. They are used for collaborative work on any Git repositories and to interact with any repository hosting site/servers, which can be GitHub, but also GitLab, Bitbucket or a self-set-up bare Git repository on a web server.

For simplicity, we will use GitHub as an example to explain commands that are used for interacting with Git repositories.

Create a Local Copy of an Online Repository

So far, all Git commands introduced in this chapter are concerned with local, unconnected Git repositories. In order to collaborate with others, hosting services, such as GitHub, can

store a *clone* (a copy) of your local repository and expose it to others. Usually, you will have a local repository and a *remote*, web-hosted repository. Your local repository is connected to the web-based clone. In technical terms, the web-based clone is a `remote` of the local repository. Usually, this remote is called "origin". Having a web-based remote allows you to *push* changes to your project online. It enables others to obtain their own clone of your repository (a copy of your repository to their local computer), make changes, and submit a *pull request* that allows you to integrate their changes. For example, one can create an independent local copy of a project using the following Git command:

```
git clone <insert GitHub link of the repository here>
```

Collaborators can update their local version of an online repository or *pull* other's work into their copy using the command:

```
git pull
```

Similarly, they can edit files locally and stage their updates (`git add .`), commit changes to a new version (`git commit`) and *push* changes to the remote online repository using the Git command:

```
git push
```

Link a Local Project on Your Computer to an Online Repository

To link a project on your computer to a new GitHub repository (preferably with the same name), you need to follow the standard workflow for creating a Git repository (described in the General Workflow subchapter) by issuing the following set of commands in the terminal, one by one:

```
cd <your project folder>
git init
git add .
git commit
```

Assuming that you have a GitHub repository that you want to connect with this project, run the following command:

```
git remote add origin <GitHub repository link for your project>
```

Then, *push* all the files on your computer to the online version so they match:

```
git push -u origin main
```

You can then go on and make more commits on your computer. When you want to push them to your online version, similarly you do:

```
git push origin branch_you_want_to_push_to
```

You can also make changes directly on GitHub by editing the online repository, and *pull* those changes locally by using the `git pull` command.

Others can also clone the repository to their computer by using:

```
git clone git@github.com:your-github-username/repository_name
```

They can make and commit changes to the code without impacting the original, and push their changes to *their* online GitHub account using:

```
git push -u origin main
```

The same procedure applies to you if you want to clone someone else's repository.

Pull Requests

If you are working on a personal branch and some other changes were made in the main branch, you can *pull* those changes down to your branch using the Git command:

```
git pull origin main
```

When everyone has a copy of the project on their own branch (checkout your branch with `git checkout branch-name`), they can *push* their changes to their branch using the following command:

```
git push origin branch-name
```

However, if you can not directly edit the repository (when you are not an owner or admin of the project), you will be able to share your work with the help of *pull requests*. A pull request allows a contributor to get the proposed changes from their branch or repository integrated into the main branch of the project. It is also possible to make pull requests via the command line (see the GitLab documentation here).

Contributing to Other Projects

When you create a local copy of a repository, you only keep the versions of the files that are in the repository at the time of creating that copy. If any changes are made in the original repository afterwards, your copy will get out of sync. This can lead to problems like conflicting file contents when making a pull request or merging changes from your branch to the main repository. Therefore, when working on different branches or forks of a repository, it is a good practice to keep them updated with the main repository and in sync with the original repository.

A Workflow to Contribute to Others Github Projects via `git`:

Using the fork button on the GitHub repository you wish to contribute to, create a copy of the repository in your account. The main repository that you forked will be referred to as the “upstream” repository.

You can now work on your copy using the command line, via the following steps (make sure you replace the placeholder user and repository names):

1. Clone it to your local machine:

```
git clone git@github.com:your-github-username/repository_name
```

2. Add the ‘upstream’ repository to the list of remote repositories using the `git remote` command:

```
git remote add upstream git@github.com:upstream-github-username/reposito
```

3. Verify the new remote ‘upstream’ repository:

```
git remote -v
```

4. Update your fork with the latest upstream changes, by first fetching the upstream repository's branches and latest commits to bring them into your repository:

```
git fetch upstream
```

5. View all branches, including those from upstream:

```
git branch -va
```

Make sure that you are on your main branch locally, if not, then checkout your main branch using the command `git checkout main`

6. Keep your fork updated by merging those commits (fetched from the upstream) to your own local main branch.

```
git merge upstream/main
```

Now, your local main branch is up-to-date with everything modified upstream. If there are no unique commits on the local main branch, git will simply perform a fast-forward.

Note: The upstream/main is the original repository's main which you wish to contribute to, whereas origin/main refers to the repository you cloned in your local machine after it was forked on GitHub.

Once your fork is in sync with the upstream main repository, you can always keep your local cloned repository in sync with origin (fork in this case) by using:

```
git checkout main
git pull
```

The `git pull` command combines two other commands, `git fetch` and `git merge`. When using `git fetch`, the resulting commits are stored as the remote branch allows you to review the changes before merging.

Similarly, if you have created more branches other than main, you can also keep them in sync with your main, once it is in sync with the upstream repository.

```
git checkout my-other-branch
git pull origin main
```

When everything is up-to-date, you can work on your branch and commit changes.

When you are ready to push your local commits to your forked repository (origin), use the following command.

```
git push origin forked_repository
```

Now you can make a pull request!

Good Practice

Before you create a branch, make sure you have all the upstream changes from the origin/main branch.

A word of caution on the `rebase` command: While trying to keep your branches in sync, you may come across the `rebase` command. It tends to rewrite history and could be troublesome if not communicated with others working on the same branch. Try to avoid using the `rebase` command, and instead use `pull` or `fetch`+`merge`, as discussed in this section. You can find more details about Merging vs Rebasing.

Further reading

- An article on syncing a fork of a repository to keep it up-to-date with the upstream repository.
- Instructions if you wish to do it all in the browser itself.