DHTech Education & Training Series: Git & GitHub

Julia Damerow, Fernanda Freire

doi/1234

Git and GitHub

In this lesson, we will talk about version control with Git and GitHub. Some of you might have already heard about it or even used it. Git is a very popular version control tool mostly used by developers but it can very well be applied to non-coding projects as well! You could use it to write a paper or to manage your dataset. If you do develop code, however, you should not live without Git or another version control system. There are several other version control systems such as Mercurial and SVN but Git is the most used in research. In the international research software engineer survey from 2022, Git was used by 93.75% of all survey respondents.

But what are version control systems? Version control systems are systems used for tracking changes in files. They allow multiple users to collaborate on the same project in parallel, maintaining the integrity of the files and avoiding overwriting each other's work. They offer features such as maintaining a history of the changes, being able to revert to previous versions, and merge different versions of the same file.

Git is a piece of software originally developed by Linus Torvald that you install on your computer to version your files. GitHub, in contrast, is a cloud-based service used for hosting and managing Git repositories. It provides a series of collaborating tools that allow for different access permissions for collaborators, issue tracking, feature requests, documentation, and several other features.

If you want to know more about why using Git and GitHub is useful, check this page.

To start out, you should first work through one or both of the following Git tutorials.

- Software Carpentries Git Tutorial: Version Control with Git
  This tutorial is fairly detailed in the descriptions and walks you through an example project with Wolfman and Dracula.
- Code Refinery Git Tutorial: Introduction to version control with Git
  This tutorial offers a hands-on approach to learning how to work with Git, with exercises and examples but with a little less text.

Getting used to working with Git and Github can be a bit confusing in the beginning, but don't be scared: in our FAQ section, we selected a series of common questions and problems you might have in your journey alongside short explanations and useful links. If you have questions that are not included in our FAQ, open an issue in our repository so that we can include it.

Git and GitHub work with a specific vocabulary to describe its functionalities and processes. In the next section below you can find a short description of the most common actions involved in the version control workflow of Git and GitHub.

Git Terminology

There are certain terms you will need to know when using Git to effectively talk to other people who use Git and to understand Git's documentation and Git tutorials. This section will explain the most important terms.

Clone

In version control, to clone is an action used to make a copy (or clone) of an existing repository while maintaining a reference to the original (copied) repository. The original repository is typically in the cloud (it's called the *remote*). The copied repository is usually on your local computer.

Commit

In version control, to commit is an action used to record the current state of a repository and effectively create a new version of your files. You can go back to commit after you made further changes. So if you made changes that you realize you don't want to keep, you revert your files to a previous commit.

Stage

In version control, to stage is an action that selects a group of changes to be recorded through a commit. This means that not all changes that you perform on your files are automatically included, and therefore recorded, in a new commit. You explictely have to tell Git what changes you want to include in your next commit by *staging* them.

Push

When working locally on a copy of a repository hosted remotely, the changes that are carried out and commited need to be sent from the local copy to the remote repository in order to be applied. This process of transfering the commits from a local to a remote repository is called push.

Pull

When several collaborators are working together on a repository hosted remotely, it is very likely that they will work locally on copies of the repository and regularly transfer their commits to the remote repository through pushes. What happens when the remote repository goes through changes implemented by another user? You need to update your local copy to include these changes. The process of downloading the content of a remote repository and immediately updating the local repository to match the updated content is called pull.

Fetch

To fetch is an action that downloads the content of a remote repository while not integrating any new or changed content into your local copy of that repository. Fetching a remote repository gives the user an overview of the implemented changes that this repository has undergone while not directly implementing them to the local data.

Git Workflows

When you start working with a Git repository, especially when you work with multiple people but even if you are working alone, you might start to wonder how to most effectively work with branches. When do you create a new branch? When do you merge? How many branches should you create? Well, you're not the first to think about this! There are several widely-used workflows that provide guidance on these questions. You can find a brief description of several workflows on this GitLab page.

Generally, it is good to remember that the more short-lived your branches are, the easier it will be to merge it back into the main branch. Some workflows even completely avoid using branches with the goal to avoid messy merge situations.

Git Forking

You might have noticed at some point that there is a button on a GitHub repository page that says "Fork". It looks something like this:

When you click on this button, GitHub will create a copy of the repository into your own account. This *fork* will have all the content that the original repository has, but you can change the files whatever you like, it will not affect the files in the original repository. You can, however, request to merge the changes you make to your fork back into the original repository.

Forking is a very powerful tool and a lot of open source software depends on it. It allows you to use the code from another repository, modify it, and even go back in the history of the code without changing anything in the original repositories. There is a Git workflow build around forking that many open source projects use to allow contributions from developers not part of the core development team.

FAQs

In this section, you will find a running list of questions that come up frequently. If you have a question that is not answered here, feel free to open an issue in our repository.

What's the difference between fetch and pull?

When you fetch a branch, you copy the changes that exist in the remote repository into your local repository. However, you will not see these changes when you look at the files because the changes will only be in your local repository not in your working directory. To see the latest changes, you will need to pull.

What does it mean to force push?

Force pushing is an action you should avoid whenever possible. *Do not do it unless you know exactly what you are doing!* When you try to push a commit you made locally, Git will sometimes tell you that it can't execute the push (for various reasons). One solution you will find online to this problem is that you should force push. However, this is very dangerous! When Git refuses to let you push, it means that for some reason the remote repository that you try to push to and your local repository are out of sync. For instance, this can happen when you work with someone else, and they push some changes since you last pulled. In this case there will be an extra commit in the remote repository that you don't have, and you will have a commit that is not present on the remote. If you would force push in this scenario, you would tell Git that you don't care and that you just want it to override the change in the remote repository, effectively deleting the work of your colleague.

What is *stashing* in Git?

When you stash your changes, you basically save them locally without committing them with the goal to get a *clean* working directory (a working directory in which your files do not have any pending changes). You sometimes want to do that when you made changes on one branch and are not ready to commit them yet, but need switch to a different branch temporarily. In that case you can stash your

changes, switch to another branch, work on the other branch and then come back to your first branch (or another!) and get the saved changes back from the stash (called *applying* the stashed changes).

Resources

- Wikipedia entry about Git
- Software Carpentries Git Tutorial
- Code Refinery Git Tutorial
- Git Workflows