How to use Git and GitHub

J-F-B-M *

December 10, 2014

Contents

Abstract

This Document is directed towards everyone who just started with **GIT**. It will explain you how to get a repository to your computer, and how to get your changes back to the server.

*alias Larethian

1 Introduction

So you found our project and thought about the possibility to dive into it, but soon you noticed that there is this **GIT**. In this paper, I want to introduce you to **GIT**, showing you how and why to use it.

2 What is GIT?

GITis a distributed version control system. Don't worry, that simply means that multiple persons can work at the same documents either at the same or at different times without breaking everything for each other. To effectively collaborate in this project, you **HAVE** to install **GIT**and get a **GitHub**-Account. You can do everything via **GitHub**'s Web-interface, but doing so is tedious and unproductive.
To install **GIT**, visit the **GIT**-Homepage. There is also a good explanation about the Git Basics.
GITis organised in repositories. One repository can not look into another[1]. Think of it as different folders. You can clone, pull and push repositories, but more on that later.

3 What is GitHub?

GitHubis a free hosting service for repositories. It provides us with many tools that come quite in handy for our goals, so that's why we use it. And because it's free.

[1]That is not true. But we limit our view to this to get you started in the wonderful world of **GIT**.

4 How to use GITand GitHub?

First, you have to get a **GitHub**-Account. Once you have this, tell your name to us in our Chat, and we will add you to our organization as soon as possible. Then you have to install **GIT**. **GIT**is in itself a program like every other program, but you can't use it directly. You either have to use another program which gives you a GUI, or use a command-line. As this document can't cover every possible GUI, I will just show you the command-line commands.

`cd <PATH>` This will change the current directory of your command-line. As your local repository will be created in the directory you are currently in with your command-line, you might change it with this command.

`git clone <URL>` You do this **once** at the start. This command will copy the repository from the server to your local computer[2].

`git pull` This will pull all changes on the server to your computer. If you haven't done anything in your copy, the server and you will have the same data after this command. The `clone`-command does something similar to `pull`, but also some additional work.

`git add <FILE>` With this command, you can add new files to the supervision of **GIT**or tell git that you want to commit changes to existing files. Normally, when you create a new file, **GIT**will ignore it. You have to tell **GIT**that you want to track this file. To simply add everything, you can type `git add -A`.

`git commit` With this command, you can create a complete message containing all your (added) changes. A command-line-text-editor will open and you have to enter a message, which should summarize your changes. With `git clone -a` you can automatically add all changes to already known files to your commit, but you can't add new files. With `git clone -m "Some text"` you can directly add a message to your commit, skipping the text-editor. You can also combine this to `git commit -a -m "Some text"`.

`git push` This will push all your commits to the server. Until this command, only your local copy has been modified. After this command, your changes are online and for everyone to see. You will be asked to enter your user name and password. You want see any stars as usual when entering your password. Don't wonder, your password is still entered. There are no stars shown to make it harder for others to see your password-length when they *accidentally* pass by your desktop. Just keep typing.

A normal **GIT**-session will look like this:

1. `git pull` See what the others did while you were away.
2. Do some work.

[2]If you add your user name in the URL, you don't have to enter it every time you push. Example: `git clone https://J-F-B-M@<URL without https://>`

3. `git add -A` Add all your changes to the next commit. Alternatively add only special files.
4. `git commit` Enter a message describing what you did.
5. `git push` Enter your User name and Password.

You see, it's simple enough.

5 What to do when everything goes wrong?

There are many things that can go wrong with **GIT**. I work with **GIT**for nearly 4 years now and even I make a lot of mistakes. Luckily **GIT**is very friendly and lets you undo a lot. To really screw up, you have to enter so many wrong commands that you should already be worried by then or you want to harm the project, which I happily believe you don't want.

5.1 I accidentally added a file to the index!

Sometimes you added a file to the data that should be committed, and realized afterwards that you don't want it in this commit.
To undo this, simply do
`git reset HEAD <File>`
where `<File>` is the file you don't want to commit.

If `<File>` is in a subdirectory (not in the same folder your commandline is in), you need to give the path from your current directory to the directory containing the file. A linux-commandline could look like this:
`user@computer /Documents/MyWorld $ git reset HEAD ./path/to/file.txt`
This will reset only the file at `/Documents/MyWorld/path/to/file.txt`, but would leave anything else untouched.

To reset every file you can use the following:
`git reset HEAD .`
This will reset all files in the same directory and all subdirectories. You might need to navigate around using `cd <path>` to get to the right directory.