Munich
Germany
Never up to date

Report

Git - Basics and More

General Use Cases and Background Knowledge

Paul Heidenreich

November 20, 2021

Preface

This is the preface.

Contents

List of Figures

Abbreviations and Acronyms

e.g.	for example
i.e.	that is to say

1 Cloning and Creating Git Projects

This chapter describes the workaround to clone git repositories from a remote server or network as well as creating a brand new local git project on your machine, for example (e.g.) when you start your own new software project. Additionally, the proceeding of creating a remote repository of your newly created local git project on GitHub is presented.

1.1 Cloning a Git Repository from Remote

Cloning a git repository is straight forward. Open your terminal or command line and navigate to the location where you want to place the project on your computer's file system.
Before executing the famous **git clone** command take a short breath and ask yourself: "Does this git project has sub-modules and do I also want to clone them too?"
If you say yes of course, then execute in your terminal/command line

```
$ git clone --recurse-submodules <Path/URL/SSH of Git Project>
```

Otherwise, the **git clone** command simplifies to

```
$ git clone <Path/URL/SSH of Git Project>
```

In case, you did not know that the project you already cloned does contain submodules or you simply forgot to clone them in a hurry, there is absolutely no reason to get into panic;)
Keep clam, grab your keyboard and type

```
$ git submodule update --init --recursive
```

Here, the important thing to learn is to use the git way to solve problems. Do not get the idea of just deleting the newly cloned repository and clone it again with the right command!

User Story: *"How do I know if the project I want to clone has any submodules integrated?"*

The first option is to always clone the project with the recurse flag, that is to say (i.e.) submodules get cloned too:

```
$ git clone --recurse-submodules <Path/URL/SSH>
```

The second option is to clone the main project without the recurse flag and checking for the existence of any submodules afterwards by

```
$ git clone <Path/URL/SSH>
$ git submodule status
```

As a result you see all submodules of the project currently missing (there is a little minus sign infront of the SHA key). Clone these submodules into your project repository by executing

```
$ git submodule update --init --recursive
```

1.2 Creating a Local Git Repository inside a Project Folder

Each new git project starts with the **git init** command.

User Story: *"Create a new, local git repository on your machine for the project you want to start or for the one you already started, i.e. you have already produced some files and want to organize your working progress with git."*

In this case, navigate into the directory in your file system which will contain the top level resources of your project or where you have saved the your work done so far. Inside this directory execute

```
$ git init
```

As a result, your directory now contains a **.git** repository which contains everything needed to version your project and track your changes over time. Briefly said, the top level project folder now consists of two main parts:

1. the project's actual working tree, i.e. all the files, code and other work inside this folder and second,

2. the git repository which owns all the snapshots of your work that you put inside this folder at times (the so called commits). This is the **.git folder** inside your project folder created with the **git init** command.

Now, everything is ready to use all the other git functionality.

1.3 Adding a Remote Repository to an Existing Local Git Project

1.3.1 Local and Remote Git Repositories

Before walking through this little workaround, it is advisable to recall oneself into mind the main difference between a local and remote git repository.
As explained in Section 1.2 a local git repository consists of the project's working tree and the

git "archive" (.git folder) recording all the development and changes made so for. By contrast, a remote git repository does only consist of a copy of your local git "archive" made at a time. There is no working tree inside a remote git repository. However, it is possible, via **git clone**, see Section 1.1, to locally reproduce a working tree from this **.git** archive.
Summarizing it all up:

$$\begin{aligned} \mathit{Git\ local\ Repo} &= \mathit{Working\ Tree} + .\mathit{git} \\ \mathit{Git\ remote\ Repo} &= .\mathit{git} \end{aligned}$$

A git remote repository, i.e. a git repository with no working tree, is also called a **bare** repository. Please note, that projects on GitHub are no standard, bare remote repositories. Actually, they are checked out projects belonging to a real remote .git folder somewhere on a server. Interacting with remote repositories containing a working tree can lead to strange behaviorism, see Section 1.3.2 as an example.

Let us have a closer look at this **.git** folder. In fact, the **.git** "archive" rally consists of two main git features. There is, on the one hand

- the **Staging Area** or **Index** and on the other hand we have
- the **git Commit Tree**.

Putting it all together, a whole local git project consists of the **Working Tree** plus the git **Staging Area/Index** plus the **Commit Tree**!
In analogy to the the famous OpenGL graphics pipeline, I call these three stages the **git Pipeline**. Since data is pumped from the Working Tree over the Index into the Commit Tree, this comparison seems valid. Chapter 2 introduces and describes the journey through the pipeline in detail. Here, the main thing to remember is, that a local git project has a working tree whereas a remote one does not and the .git folder contains the Staging Area and the Commit Tree!

1.3.2 GitHub as Remote Location

User Story: *"I want to crate a remote location for my software project on GitHub (or any other Server) and connect this repository with my local one."*

Creating a remote GitHub repository requires a GitHub account! Follow the steps below:

1. Go to your GitHub homepage and click the +-icon in the navigation bar (upper right corner) and select **New Repository**.
2. In the Repository Name field, enter a short, descriptive name. I prefer using the project's local name and no white spaces.
3. Enter a brief description. This is optional, but it helps users understand the purpose of your project when it appears in search results or in your repository list.

4. Keep the default Public setting for your project's visibility. If you select Private, the community won't be able to access your project repository.

5. Leave Initialize this repository with a `README` unchecked if you have already created a `README.md` file for your project.

6. Click the Create Repository button to create the new GitHub repository.

Afterwards, the process continues on your local machine at the top level of your local git project folder. Check the status of the remote locations connected with your project with **git remote**. It everything is as expected, add the newly created GitHub remote location to the project by using the `URL` or `SSH` address provided by the GitHub page:

```
$ git remote -v
$ git remote add <Name_of_your_local_Project> <URL/SSH from GitHub>
$ git remote -v  //to see if the new remote location is listed
```

Finally, set the upstream branch. That is, tell your local git repository the name of the remote one (see the second item when creating a remote GitHub project) and the commit-branch to which you want to push back new commits. In General, this branch is called master or main (by default):

```
$ git push --set-upstream <Remote_Repo_Name>
<master/other_branch_name>
```

Now, you are ready to push if there are unpublished, new commits:

```
$ git push
```

Sometimes, setting an upstream branch and pushing your local contents for the first time to GitHub may cause error messages like *fatal: refusing to merge unrelated histories* or *git error: failed to push some refs to remote.* Most of the time, this is due to adding files or some other new content directly on the GitHub web page. A **README.md** file for example. Then, we have actually two separate projects, the local one on your machine and the "local" one on the GitHub web page which started their lives independently from each other but refer to the same remote .git folder. The task is now to tell git, that these two projects are actually one and the same with no risk to enable a merge/pull or push request (in our case pushing local content to GitHub and pulling the README.md file created there). Of course, the git community has already encountered this problem. By executing

```
$ git pull <Project_Name> <remote_branch> --allow-unrealted-histories
```

where the remote branch is master or main you tell git that both projects are the same and pulling the README.md file is safe. See [2] for more information.
However, be very careful with this command and use it only if you definitely know what your are doing!!!

1.3.3 An Arbitrary Server as Remote Location

This is almost the same procedure as described in Section 1.3.1. The only difference is, that there exists probably no web page with a graphical GUI interface, like GitHub, that leads one through the process of creating a remote repository. But no panic, it becomes almost simpler!

Navigate on the server's file system or to the network drive where you want to generate your remote project destination. Then create a **bare** git remote repository (i.e. a repository without a working tree) with

```
$ git init --bare
```

Afterwards follow the steps already described in Section 1.3.1:

```
$ git remote -v
$ git remote add <Name_of_your_local_Project> <Path_to_remotre_repo/
  URL_to_server>
$ git remote -v
$ git --set-upstream <Remote_Repo_Name> <master/other_branch_name>
```

1.4 SSH (Secure Shell) Keys

SSH (Secure Shell) are very practical to communicate with your remote location safely and without typing the password of your account each time you communicate with your remote server. You can create a SSH key to your GitHub account (where your remote git repositories are) and connect your local git project via a SSH key instead of the URL. Therefore, you have to create an SSH key on GitHub and register your local projects with this key.
This section is not about SSH keys in detail, only how to use them in the git context.

User Story: *"I want to replace the remote URL with an SSH key."*

First thing to do is to list the remote repository connected with your local project and in a second step, replace the URL to your remotes with the SSH key

```
$ git remote -v
$ git remote set-url origin <ssh-key, e.g. from GitHub>
```

Control the result, again with

```
$ git remote -v
```

Eventually, if e.g. **git push** does not work without typing the password, you have to add the SSH key and activate it. In a terminal or command line type

```
$ ssh-add
$ ssh-add-l (to verify everything)
```

On Windows systems make sure that the ssh-command is on your PATH variable so that it works in a CMD (for more Information on this google for "adding ssh to path variable").

2 The (local) Git Pipeline

As shortly mentioned in Section 1.3.1 the three major parts of your local git project are

- the Working Tree or all the files and code you actually write,
- the Staging Area or Index where you prepare new content and changes of your work ready to commit and
- the Commit Tree resembling the whole history and development of your project.

These three items make up the stages of the git pipeline which, see xxx versions your whole project in an ongoing process. That is, you travel through this pipeline again and again. Most of the time the pipe serves for creating new commits. To achieve this, one has to travel the pipe from the Working Tree over the Index to the Commit Tree. However, there are occasions where it is helpful to recreate a past state of your project (e.g. you want to undo some weird code you have just committed or you want to compare an old status of your work with a new behavior). No problem, use the git pipeline to load an arbitrary commit from the git Commit Tree and recreate the underlying Working Tree represented.

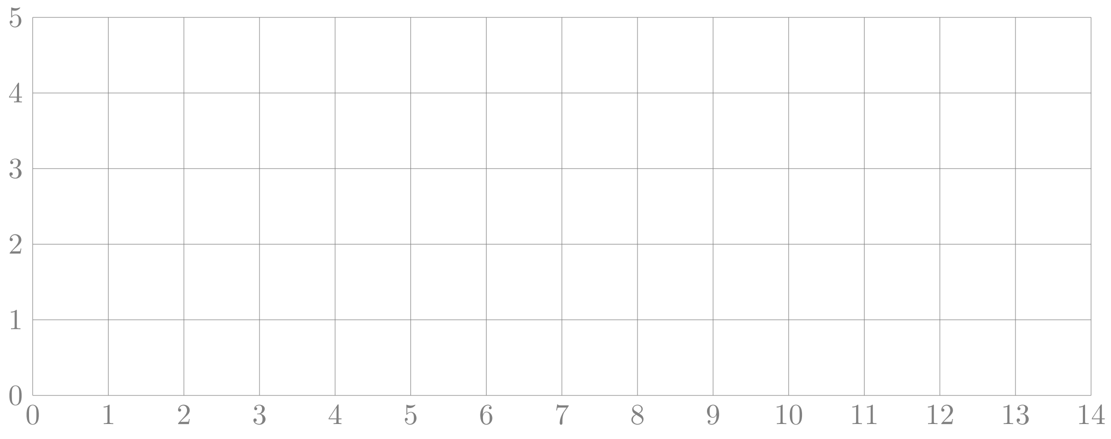

Fig. 2.1: The Git Pipeline: Working Tree, Staging Area and Commit Tree

In other words, the pipeline is the tool or vehicle to append a new commit to the Commit Tree and to reload an already existing one.

2.1 Commit, HEAD and Branch

2.2 Traveling the Pipeline Forward

2.3 Traveling the Pipeline Backwards

Hello World!

Finally, an example of how to add literature into the written text [3].

It is also possible to specify pages with [see 1, page 127].

References

[1] Tilo Arens et al. *Mathematik*. Springer-Verlag, 2015.

[2] Git. *git-merge - Join two or more development histories together*. Nov. 20, 2021. URL: `https://git-scm.com/docs/git-merge`.

[3] Scott Meyers. *Effective C++: 55 specific ways to improve your programs and designs*. Pearson Education, 2005.