A *Git* Tutorial

sunlu.electric@gmail.com

December 7, 2021

Git is a distributed version-control system for tracking changes in source code during software development, and it can be used in cross platforms including Windows OS, Unix/Linux and MacOS. This tutorial introduces the basic use of *Git* on a local Linux machine and on a remote host such as *GitHub*.

Contents

1 Introduction to *Git* and *GitHub*

Git, created by Linus Trovalds in 2005, is a distributed version-control system for tracking changes in source code and files during software development. It is designed for coordinating work among programmers and its goals include speed, data integrity and support for distributed non-linear work flows. *Git* is a free and open-source piese of software under GNU general public license V2.

As with most other distributed version-control systems, every *Git* directory on every local computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server [1]. A demonstration of how *Git* utilizes master and slave repositories (also known as “branches”) to coordinate projects is given in Fig. 1. A “master repository” is used as a shared repository where everyone pull the updated version of the project, make modifications, then push back the modified project. Each branch, when pulled, is a duplication of the master repository thus contains the entire project files.

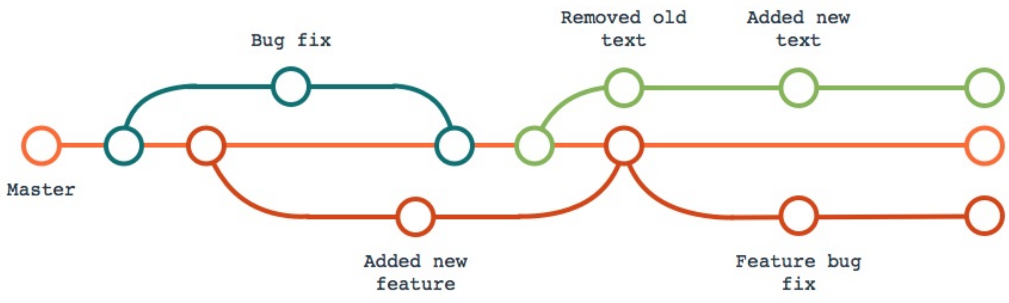

Figure 1: A demonstration of how *Git* repositories work to coordinate programmers code [2].

GitHub is a global company that provides hosting for software development version control using *Git*. It offers all of the distributed version control and source code management functionality of *Git* and also adds its own features. It provides access control and several collaboration features such as bug racking, task management and wikis for each project [3]. *GitHub* offers plans for free, professional and enterprise accounts. It offers unlimited private repositories to all the plans, and has become the largest host of source code in the world as of May 2019.

Notice that *GitHub* is not the only company that provides this kind of services. Other companies, for example *GitLab*, also provide similar services.

2 *Git* on a Local Linux Machine

2.1 Install *Git*

Git is a build-in software in most (if not all) Linux distributions. The following commands in shell should download and install *Git* on a Linux machine.

```
$ sudo apt install git
```

The user name and email can be configured as follows. Notice that this configuration is global to all *Git* projects in the machine.

```
$ git config --global user.name 'user_name'
$ git config --global user.email 'user_email@user_domain'
```

2.2 Create a *Git* Project and Add/Remove Files

Consider creating a *Git* project and work on it on a local machine. As a first step, create an empty folder for the project. Navigate to the project folder. Use the following bash command to initialize a *Git* project repository.

```
$ git init
```

By doing this, the project is initialized and further *Git* commands can be applied to it. Behind the screen, a hidden folder `.git` has been automatically created in the project folder, containing the metadata of the project. It is not necessary to manually access this hidden folder.

In each step of the project development, always use the following command to check the *Git* status, including the project files status information and versions information.

```
$ git status
```

With the current directory initialized by `git status`, the following commands add a file that already exists in the directory to *Git*.

```
$ git add <file_name>
```

The process of adding a file to *Git* is considered as a change, thus leaving these files in the staging area, whose concept is illustrated by Fig. 2. The file needs to be committed later on before the change is actually confirmed. For a file that is in the project folder but not added to *Git*, it is considered as untracked file. Use `git status` to check the status of all files in the project folder.

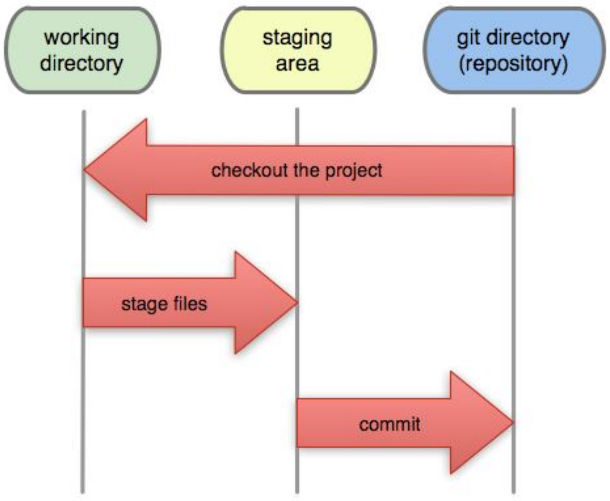

Figure 2: A demonstration of how staging area works [4].

To remove a file from *Git*, use the following commands. The removed file will then appear in the untracked file list.

```
$ git rm --cached <file_name>
```

Regular expression can be used in the above commands.

2.3 Commit a Project File in the Staging Area

The existing files in *Git* can be modified with any file editor. When changes are made to such files, they will be automatically put into the staging area (just like when a new file is added to *Git*).

Using the following command to commit all files in the staging area. A *vim* editor will pop up and ask the user to put in comments to the commit. The comments are mandatory.

```
$ git commit
```

When committing the files, the name and email information of the author and committer will be recorded. The information is taken from environment variables if set the following 6 variables.

- GIT_AUTHOR_NAME
- GIT_AUTHOR_EMAIL
- GIT_AUTHOR_DATE
- GIT_COMMITTER_NAME
- GIT_COMMITTER_EMAIL
- GIT_COMMITTER_DATE

If they are not set, the information is taken from **user.name** and **user.email**, or **/etc/mailname**, or **$HOSTNAME** of the machine.

There are some commonly used options for the commit command.

```
$ git commit [-a]                   # Commit all files
             [<file_name>]          # Commit a particular file
             [--author=<author>]    # Override the commit author
             [-m <message>]         # One line comment, instead of vim comment
```

File *.gitignore* in the directory can be used as a ignore list for *Git*. Files and directories that are added into *.gitignore* will not be monitored or tracked by *Git*. However, do notice that *.gitignore* will be monitored and tracked by *Git*. Regular expressions can be used in *.gitignore* file as well. This becomes handy when there is a log file in the directory which does not necessarily need to be tracked.

2.4 Work on Different Branches

By default, when a project is created, the user works on the Master branch. This is reflected when using `git status`, where it is shown *"On branch Master"*. Therefore, all the commit mentioned in the previous section is done on Master branch.

In practice, it is more often that a separate branch is created for a particular upgrade or modification purpose. Only when the functions in the separate branch has been tested, would it be merged to the master branch. This is shown in Fig. 1.

Using the following command to create a branch. Notice that this does not change the current branch automatically.

```
$ git branch <branch_name>
```

Use the following command to switch branch.

```
$ git checkout <branch_name>
```

At this moment, all commit made using `git commit` will be done on the switched new branch. All the changes made on this branch, including the changes in the staging area and yet not committed, are transparent to other branches including the Master branch.

The new branch can be merged to the master as follows. First, switch back to master branch. Then, use the following command. A *vim* editor will pop up and ask the user to put in comments to the commit. The comments are mandatory.

```
$ git merge <branch_name>
```

This create-switch-merge branches approach is handy for version control especially in collaborative projects.

3 *Git* on Remote Repository (Such as *GitHub*)

3.1 Register a Remote Repository

To use *GitHub*, the user needs to register a *GitHub* account and create a repository. Then the *GitHub* repository can be added to the local machine as a remote repository as follows.

```
$ git remote add <remote_name> <remote_url> # Add remote repository
$ git remote rename <old_remote_name> <remote_name> # Change remote_name
$ git remote set-rul <remote_name> <new_remote_url> # Change remote_url
$ git remote remove <remote_name> # Remove remote repository
```

For example,

```
$ git remote add raspi4b https://github.com/sunluelectric/raspi4b.git
```

Notice that the remote name is not necessarily the same with the repository name in *GitHub*. It is simply the name by which the local *Git* is tracking the access point. Use `git remote -v` to check all remote access points stored in the local machine.

Authentication is required when the user interact with the remote repository from the local machine.

The use of the above method to register a remote repository is not limited to *GitHub*. The same method can be applied to other remote *Git* management system.

3.2 Download a Clone of a Project

Use `git clone` to download an existing repository from remote repository as follows. A new directory is created on the local machine for this repository.

```
$ git clone <repository_url> [<directory>]
```

When using `git clone`, the repository is automatically added into `git remote`, and all branches change on the remote repository are tracked. If changes, for example updated, are made to the remote repository, `git pull` can be used to pull the changes.

3.3 Pull Updates and Push Changes from/to Remote Repository

Use `git pull` to download the latest version of a branch in the repository. It is configurable what to be carried out when the existing local versions conflicts with the existing remote versions.

```
$ git pull [<options>] [<repository>]
```

Use `git push` to upload a file, or more often, a branch, into the remote repository. Usually, the changes made in the local machine has already been committed to a branch and `git push` will upload the updated branch to the remote repository.

```
$ git push [<options>] [--repo=<repository>]
```

The authentication is required when pushing changes to the remote repository.

References

[1] Wikipedia. Git. [Online]. Available: https://en.wikipedia.org/wiki/Git

[2] S. K. Git version control series: What is git? [Online]. Available: https://blog.cpanel.com/git-version-control-series-what-is-git/

[3] Wikipedia. GitHub. [Online]. Available: https://en.wikipedia.org/wiki/GitHub

[4] Rook. What does 'stage' mean in git? [Online]. Available: https://softwareengineering.stackexchange.com/questions/119782/what-does-stage-mean-in-git