A basic approach to Git

Tran Thi Thuy-20173394
Mac Van Thiem-20173385
Mai Xuan Thang-20173369

Hanoi University of Science and Technology

June 2020

Contents

Abstract

In this article, we will learn about Git – an Distributed Version Control System, which we will begin to learn from the most basic concepts on Git, how Git works, and practice using Git with some of the basic commands introduced herein. Through this article, we can better understand Git: the concept, how it works, uses, thereby drawing out the benefits that Git brings as well as some of its limitations.

1 Introduction

Real life projects generally have multiple developers working in parallel. So a version control system like Git is needed to ensure there are no code conflicts between the developers.

Additionally, the requirements in such projects change often. So a version control system allows developers to revert and go back to an older version of the code.

We will learn about the concepts of distributed version control system, Git, how Git works, and some basic commands to get started with Git.

In order for the practice to be effective, you should prepare a computer with git installed. If you don't know how to install it, don't worry, there's a tutorial below!

2 Background

2.1 Some of the definitions about Git basic

2.1.1 Overview about version control system (VCS)

VCS is a system that records and saves the change of files in time. Thanks to it, a file could recover over the previous version. Other hands, you can follow the change of file in time, who make those changes, when it was changed,. . . there are some systems such as Concurrent Versions System, Subversion, git, Mercurial [2].

Two classes of VCS:

- Centralized Version Control System: "It stores a master copy of files history and in order to read, retrieved, commit new changes to a certain versions, the clients need to contact a server" [2].

- Distributed Version Control System:" DVCS is designed to act in both ways as it stores the entire history of the files on each machine locally and it can also sync the local changes made by the user back to the server whenever required, so that the changes can be shared with the whole team" [2].

2.1.2 What is Git? [6]

Git is a Distributed Version Control System, focuses on speed, simplicity, distribution, fitting with all of the projects having different sizes. Git is designed and developed to develop the Linux core. Git is a free system and distributed following the policy of GNU version 2 [4].

2.1.3 The pros of Git

- It is free and open source: Git is published under the GPL's open source license. It is available and free on the internet. You can use it to manage your projects without any money. As a free open source, you can download and make some changes for your command [2].
- Fast and small: Almost operations are executed in local, which brings a great benefit to speed. Git is not based on the Server, which is the reason that all operations don't need to interact with the remote server (Git Server). The core of Git is written in C, which makes less time cost than the high-level languages. Git save entire repository but the size of data is small, so, you can see its strength in compressing and storing data [2].
- Hidden backup: The data in Git was rarely lost because if Git server has trouble, it can restore data from the client [2].
- Security: Git uses a common cryptographic hash function called secure hash function (SHA1), to named and identified objects in the database. Each file and commit is checked and retrieved by the checksum. Therefore, changing any data in the database without knowing well about git is impossible [2].
- Not need strong hardware: The central server in CVCS has to be strong enough to serve to require of all clients. If the number of clients is very large, the limited ability of hardware could decline productivity.

However, Git (a DVCS) doesn't need very strong hardware because all of operations are executed in clients and there is the interaction with Git Server only when clients want to publish the changes [2].

- Branching easily: If you create a new branch in CVCS, the system will copy all of the data to the new branch, which wastes so much time and is complicated. But with Git, you only take some seconds to create, delete, and merge branch [2].

2.1.4 The elements of Git

- Repository: this is the place contains source code, detailed information, history, content change, which each client impact. There are two classifications of the repository [4]:
  - Local repository: this is in clients machine, it will be synced with the remote repository by Git commands
  - Remote repository: this is in dedicated server like GitHub, GitLab,...
- Working Directory and staging area or Index: The working directory is the place where files are checked out. When you commit an operation, Git finds files in the staging area and only these files may be committed, not all of the files [4].
- Blobs (Binary Large Object): Each version of a file is represented by a blob. One blob which is named as SHA1 hash of one file contains that file data but not contain any metadata about it [4].
- Commits: This operation is to save the present status system, history, insert, delete, update file, or folder in the repository. When executing committing, repository record the differences between this committing and the previous [4].
- Branches and checkout:
  - Checkout is that create a new branch from one branch. The operations in branching will be saved in a new branch and will not affect old branches. Each repository can contain many independent branches. The default branch is master [4].

  - The local repository contains local branches, and the remote repositories contain remote branches [4].

- Clone: copy an available repository to local [4].
- Pull: update the change to the local repository [4].
- Push: bring the change committed to one branch in the remote repository in order to everyone in the team can see and sync with [4].
- Merge and rebase: Merging one developmental branch or merging change history into another [4].

2.1.5 Git's life circle

Git's life circle starts from clone operation and the next operations are followed the threads [5]:

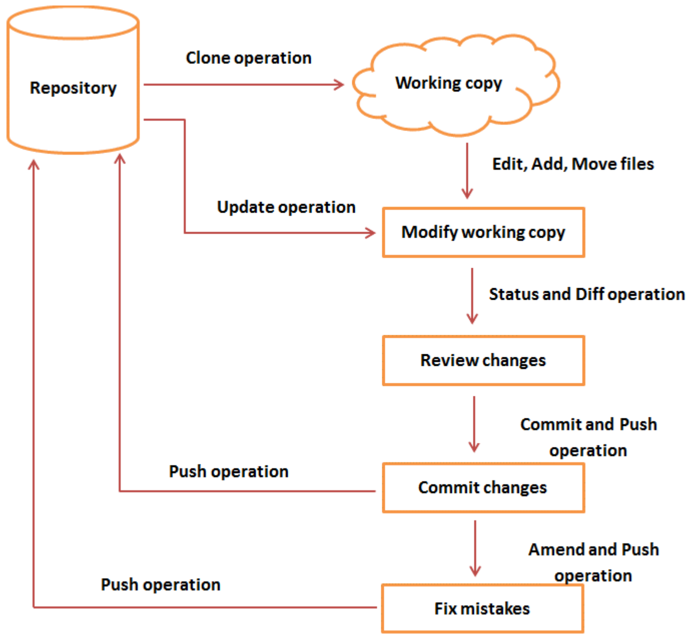

Figure 1: Git's life circle diagram

2.2 Some basic Git commands

2.2.1 Git configuration

- Set the name that will be attached to your commits and tags.
  $ git config –global user.name "Your Name" [3]

- Set the email address that will be attached to your commits and tags.
  $ git config –global user.email "youremail@example.com" [3]

- Enable some colorization of Git output.
  $ git config –global color.ui auto" [3]

2.2.2 Starting a project

- Create new local repository. If [project name] is provided, Git will create a new directory named [project name] and will initialize a repository inside it. If [project name] is not provided, then a new repository is initialized in current directory.
  $ git init [project name] [3]

- Downloads a project with entire history from remote repository.
  $ git clone [project url] [3]

2.2.3 Day-to-day work

- See the status of your work. New, staged, modified files. Current branch.
  $ git status [3]

- Show changes between working directory and staging area.
  $ git diff [file] [3]

- Show changes between staging area and index (repository committed status).
  $ git diff –staged [file] [3]

- Discard changes in working directory. This operation is unrecoverable.
  $ git checkout – [file] [3]

- Add a file to staging area. Use "." Instead off full file path to add all changes files from current directory down to directory tree.
  $ git add [file] [3]
- Get file back from staging area to working directory.
  $ git reset [file] [3]
- Create new commit from changes added to staging area. Commit must have a message.
  $ git commit "Your commit" [3]
- Remove file from working directory and add deletion to staging area.
  $ git rm [file] [3]
- Put your current changes into stash.
  $ git stash [3]
- Apply stored stash content into working directory, and clear stash.
  $ git stash pop [3]
- Clear stash without applying it into working directory.
  $ git stash drop [3]

2.2.4 Git branching model

- List all local branches in repository. With "-a": show all branches (with remote).
  $ git branch [-a] [3]
- Create new branch, referencing from the current working directory.
  $ git branch [name] [3]
- Switch working directory to the specified branch. With "-b": git will create the specified branch if it does not exist.
  $ git checkout [-b] [name] [3]
- Join specified [from name] branch into your current branch (the one you are on currently.
  $ git merge [from name] [3]
- Remove selected branch, if it is already merged into any other.
  $ git branch -d [name] [3]

2.2.5 Review your work

- List commit history of current branch. "-n count" limit list to last n commits.
  $ git log [-n count] [3]
- List commits that are present on current branch and not merged into ref. A ref can be e.g. a branch name or a tag name
  $ git log ref.. [3]
- List commit, that are present on ref and not merged into current branch.
  $ git log ..ref [3]
- List operations made on local repository.
  $ git reflog [3]

2.2.6 Tagging known commits

- List all tags.
  $ git tag [3]
- Create a tag reference named name for current commit. Add [commit sha] to tag a specific commit instead of current one.
  $ git tag [name] [commit sha] [3]
- Create a tag object named name for current commit.
  $ git tag -a [name] [commit sha] [3]
- Remove a tag from a local repository.
  $ git tag -d [name] [3]

2.2.7 Reverting changes

- Switch current branch to the [target reference], and leaves a difference as an uncommitted changes. When "–hard" is used, all changes are discarded.
  $ git reset [–hard] [target reference] [3]
- Create a new commit, reverting changes from the specified commit. It generates an inversion of changes.
  $ git revert [commit sha] [3]

2.2.8 Synchronizing repositories

- Fetch changes from the remote, but not update tracking branches.
  $ git fetch [remote] [3]
- Remove remote refs, that were removed from the remote repository.
  $ git fetch –prune [remote] [3]
- Fetch changes from the remote and merge current branch with its upstream.
  $ git pull [remote] [3]
- Push local changes to the remote. Use "–tags" to push tags.
  $ git push [–tags] [remote] [3]
- Push local branch to remote repository. Set its copy as an upstream.
  $ git push -u [remote] [branch] [3]

2.3 Git's weakness

Besides its great use, git still has certain disadvantages. First, the information model is complicated – and you need to know all of it. As a point of reference, consider Subversion: you have files, a working directory, a repository, versions, branches, and tags. That's pretty much everything you need to know. In fact, branches are tags, and files you already know about, so you really need to learn three new things. Versions are linear, with the odd merge. Now Git: you have files, a working tree, an index, a local repository, a remote repository, remotes (pointers to remote repositories), commits, treeishes (pointers to commits),branches, a stash… and you need to know all of it [1] .

The command line syntax is completely arbitrary and inconsistent. Some "shortcuts" are graced with top level commands: "git pull" is exactly equivalent to "git fetch" followed by "git merge". But the shortcut for "git branch" combined with "git checkout"? "git checkout -b". Specifying filenames completely changes the semantics of some commands ("git commit" ignores local, unstaged changes in foo.txt; "git commit foo.txt" doesn't). The various options of "git reset" do completely different things [1].

There is essentially no distinction between implementation detail and user interface. It's understandable that an advanced user might need to know a little about how features are implemented, to grasp subtleties about various commands. But even beginners are quickly confronted with hideous internal details [1].

A common response to complaints about Git's command line complexity is that "you don't need to use all those commands, you can use it like Subversion if that's what you really want". Git doesn't provide any useful subsets – every command soon requires another; even simple actions often require complex actions to undo or refine [1].

Most of the power of Git is aimed squarely at maintainers of codebases: people who have to merge contributions from a wide number of different sources, or who have to ensure a number of parallel development efforts result in a single, coherent, stable release. This is good. But the majority of Git users are not in this situation: they simply write code, often on a single branch for months at a time [1].

And finally, simple tasks need so many commands. If the power of Git is sophisticated branching and merging, then its weakness is the complexity of simple tasks [1].

3 Conclusion

Through this article, we hope to have provided you with the most general understanding of Git, master the commands to use Git in the most mature way.

4 Acknowledgement

Real life projects generally have multiple developers working in parallel. So a version control system like Git is needed to ensure there are no code conflicts between the developers.

Additionally, the requirements in such projects change often. So a version control system allows developers to revert and go back to an older version of the code.

Therefore, equipping yourself with a good understanding of Git as well

as the ability to use them well will help you participate in teamwork professionally and effectively.

References

[1] "10 things I hate about Git, 2012. Steve Bennett blogs." In: (). URL: `https://stevebennett.me/2012/02/24/10-things-i-hate-about-git/`.

[2] "Git - Basic Concepts - Tutorialspoint [WWW Document], n.d." In: (). URL: `https://www.tutorialspoint.com/git/git_basic_concepts.htm`.

[3] "Git documentation". In: (). URL: `https://git-scm.com/docs/git?fbclid=IwAR1o2ImUZsxMlR0_BqQtyVKtkyQeGRGQQ5mLoVgWvYWIE_tZDBl8SNpY1ek`.

[4] "Trung Tâm Tin Hc - H Khoa Hc T Nhiên [WWW Document], n.d." In: (). URL: `https://csc.edu.vn/lap-trinh-va-csdl/tin-tuc/kien-thuc-lap-trinh/Git-la-gi--Nhung-khai-niem-co-ban-khi-lam-viec-tren-Git-4133`.

[5] "Vòng i Git [WWW Document], n.d." In: (). URL: `https://quantrimang.com/vong-doi-git-157673`.

[6] "Zolkifli, N.N., Ngah, A., Deraman, A., 2018. Version Control System: A Review. Procedia Computer Science 135, 408–415." In: (). URL: `https://www.sciencedirect.com/science/article/pii/S1877050918314819?via%3Dihub`.