GIT & Github

Created by: Teodora Achkozidou

Date: May 2020

Contents

[GIT Terminology 1](#_Toc40361269)

[Git Version Conntrol System 2](#_Toc40361270)

[What is GIT 2](#_Toc40361271)

[Git quick start 3](#_Toc40361272)

[Version Control system 4](#_Toc40361273)

[Branches in Git 4](#_Toc40361274)

[Git Basics 5](#_Toc40361275)

[Comparison in Git 6](#_Toc40361276)

[Branching and merging 7](#_Toc40361277)

[GIT MERGE VS GIT REBASE 7](#_Toc40361278)

[GIT Stashing 8](#_Toc40361279)

## GIT Terminology

Git Goals and Objectives

* Repository contain: files, history, configuration managed by GIT
* Working Directory / Repository
* Tree stages of GIT (Working directory, Staging area /pro commit staging area/,Commit /GIT repository - history/)
* Remote repository (GITHUB)
* Master branch

**GIT INTRODUCTION**

What you get from VERSION CONTROL SYSTEM

* Save you from creating multiple backups of your files
* Allow multiple people to work on the same time on file
* Track changes & also who have made the changes
* Easy to switch back to folder versions as and when required
* Make you more productive

## Git Version Conntrol System

Version Control is the management of changes to documents, computer program, large websites and other collection of information

* Centralized Version Control System (CVCS)
* Distributed Version Control System (DVCS)

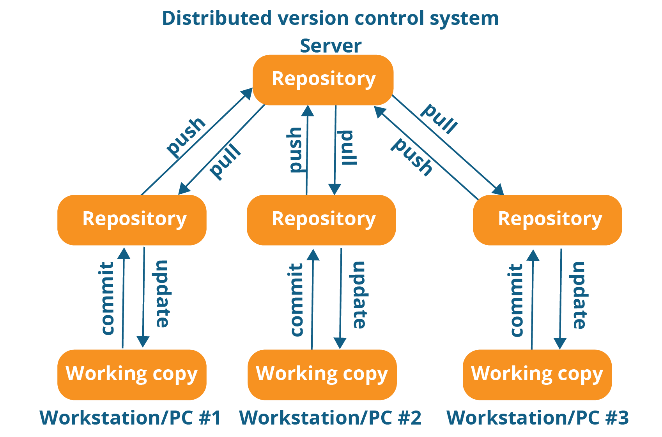
**CENTRALIZED VERSION CONTROL SYSTEM**

* A picture containing circuit

  Description automatically generatedCVCS uses a central server to store all files
* It works on a single repository to which users can directly access a central server
* The central server can be local on the remote machine directly connected to each of the programmers workstations

**DRAWBACKS OF CVCS**

* The centralized repository is not locally available
* Since everything is centralized in any case of the central server getting crash or corrupted will result in losing the entire data of project

**DISTRIBUTED VERSION CONTROL SYSTEM**

* In DVCS every contributor has a local copy or clone of the main repository
* The developer can update their local repository with new data from the central server by an operation called “pull” and affect changes to the main repository bu an operation called “push” from their local repository

**ADVANTAGE OF DVCS**

* All operations are very fast because the tool needs to access the HDD only
* Committing new change-set can be done locally without manipulating the data on the main repository
* If the central server get crashed at any point of the time, the data can be easily recovered from anyone of the contributor's local repositories

## What is GIT

The most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open-source.

Source - <https://gdgucc.blogspot.com/2017/10/what-is-git.html>

Design philosophy:

- Free and open-source

* blazingly fast
* Distributed
* Data assurance

**WHAT IS GITHUB**

We’ve established that Git is a version control system, similar to but better than the many alternatives available. So, what makes GitHub so special? Git is a command-line tool, but the center around which all things involving Git revolve is the hub—GitHub.com—where developers store their projects and network with like-minded people.

A repository (usually abbreviated to “repo”) is a location where all the files for a particular project are stored. Each project has its own repo, and you can access it with a unique URL.

Source - <https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/>

## Git quick start

* First step is to download GIT from the link below, it is available for all operation systems:

<https://git-scm.com/downloads>

* GIT Basic start configuration

$ git config --global user.name “Your Name Here”

$ git config --global user.email [“email@example.com](mailto:email@example.com)”

$ git config --list (show all properties)

$git help <attribute> (example: $git help add)

**FIRST REPO AND COMMANDS**

Enter to the folder you want to use for the project and open terminal on it

$git init

Commiting first file

$ git status

$ git add file.txt | $ git add . (add all files you made changes on)

$ git commit –m “message with wich you want to commit the file”

$ git push origin master (explanation after)

If you want to see the changes from certain user

$ git log --author ”name”

**GIT WORKFLOW**

## Version Control system

Verify changes in GIT

$ git diff

Compare staged with repo in GIT

$git diff --staged

Delete file

$git rm file.txt

GIT with Github

A screenshot of a cell phone

Description automatically generatedGIT – DVCS is a tool

GITHUB is

* Code hosting platform
* Central repository
* Repository hosting service

To create a central repository:

$ git pull origin master

$ git clone https://github.com/user/repo

$ git push origin master

## Branches in Git

* Branch in GIT is pointer to commit
* To create a new branch we use the following command:

$ git branch <branch\_name>

$ git branch development

$ git checkout development (to switch branches)

$ git branch (to check in which branch you are)

**MERGE BRANCHES IN GIT**

* Get to the branch you want to merge the changes (for example from development to master)

$ git checkout master

$ git merge development

$ git push origin master (sync with github)

## Git Basics

**REVERT COMMIT IN GIT**

$ git reset HEAD filename

(HEAD - alias for the current branch ex. $git reset master|development filename)

$ git reset HEAD~ filename

(HEAD~ last commit reverted | HEAD~5 delete the last 5 commits from the history)

The prefered command for many cases is

$ git revert commit\_id

You can coppy the commit\_id from

$ git log

Delete, rename & move file in git

$ git reset HEAD filename

$ git checkout --filename

$git rm –rf directory\

(r – recursive, f – force, all in the direcory)

$ git commit –m “message” | $ git push origin master

$ git mv filename new\_filename

$ git mv filename new\_dir\_path

Find, commit, file history

Command to verify last commit

$ git log

Get GIT abbrev commit hash

$ git log --abbrev --log

GIT oneline commit

$ git log --oneline --graph --decorate

Logs that have been executed last day or last 5 days

$ git log --since=“5 days ago”

$ git show commit\_id

$ git help log

**GIT ALIAS**

GIT alias is a short way to display a results from long command

In order to add an alias we need to add the alias at GIT global config

$ git config --global <shot\_command> “long\_command”

Example: $ git config global alias.history “log --all --graph --decorate --oneline”

or:

$ nano ~/gitconfig

**EXCLUDE UNWANTED FILES IN GIT**

$ nano .gitignore

GIT ignore pattern example:

# exclude everything except directory

/\*

!/dir

/dir/\*

!/dir/bar

## Comparison in Git

Working directory and staging directory

$ git diff

$ git fiff filename

Compare working directory and repository

$ git diff HEAD (compare working directory with last commit)

$ git diff HEAD filename

Staging area and repository

$ git diff --staged HEAD

$ git diff --staged HEAD filename

Compare commits in GIT

$ git diff commit\_it commit\_it

$ git diff HEAD HEAD^ (compare last commit and the commit previews last head -1)

## Branching and merging

|  |  |
| --- | --- |
| Verify branch  $ git branch -a  Switch GIT branches  $ git checkout <branch\_name>  Rename branch  $ git branch –m <old\_name> <new\_name>  Delete branch  $ git branch -d <branch\_name>  (before delete branch switch to another) | Merge one branch to another  $ git merge <source>  Example (first move to master):  $ git merge development |

## A screenshot of a cell phone Description automatically generatedGIT MERGE VS GIT REBASE

A close up of a map

Description automatically generated

**GIT MERGE**

* GIT merge to create a new “merge commit” in the development branch that ties together the histories of both branches, giving you a branch structure that looks like graph
* How this impact:

in this case, the development branch will have an extraneous merge commit every time you need to incorporate upstream changes. If the master is very active this can pollute your development branch history.

**GIT REBASE**

* As an alternative to merging you can rebase the development branch into the master branch

$ git checkout development\_branch

$git rebase master

* This moves the entire development branch to begin on the top of the master branch, effectively incorporating all of the new commits in the master. Instead of using merge commit, re-base re-write the project history by creating brand new commits for each commit in the original branch
* Benefits:

you get much cleaner project history, it also results in the perfectly linear project history

## GIT Stashing

**WHAT IS GIT STASH?**

* When you create a stash, you are saving uncommitted changes so that you can work on other things without losing your changes.

Example: you are working on a function, but your boss wants you to do something immediately and you need to change branches and your code is not ready for commitment and you do not want to lose your work as well… so you stash ☺

$ git stash save “message of what you were doing”

$ git stash list (list the changes made on stash with them id)

In order to work again on the stashed file:

$ git stash apply stash\_id (example: stash@{1})

STASHING: APPLY VS POP

When you’re ready to finalize these saved changes, you have two options: apply or pop.

* Apply will take the stashed changes, apply them to your working directory, and keep the changes saved as a stash.

$ git stash apply stash@{0}

* Pop will do the exact same thing for the first two steps, but it will permanently delete the stash.

$ git stash pop (will drag the very first stash on the stash list)

Source - <https://dzone.com/articles/learning-git-what-is-stashing>

**STASHING: DROP & CLEAR**

* Discard the stashed changes:

$ git stash drop stash@{0}

* Discard all stashed changes:

$ git stash clear