Contents

[What is git and GitHub? 1](#_Toc136183542)

[Git Commands 1](#_Toc136183543)

[Git Branch 2](#_Toc136183544)

[Git Stash 3](#_Toc136183545)

[Git Merge 3](#_Toc136183546)

[Git Rebase 4](#_Toc136183547)

[Git Squash 4](#_Toc136183548)

# What is git and GitHub?

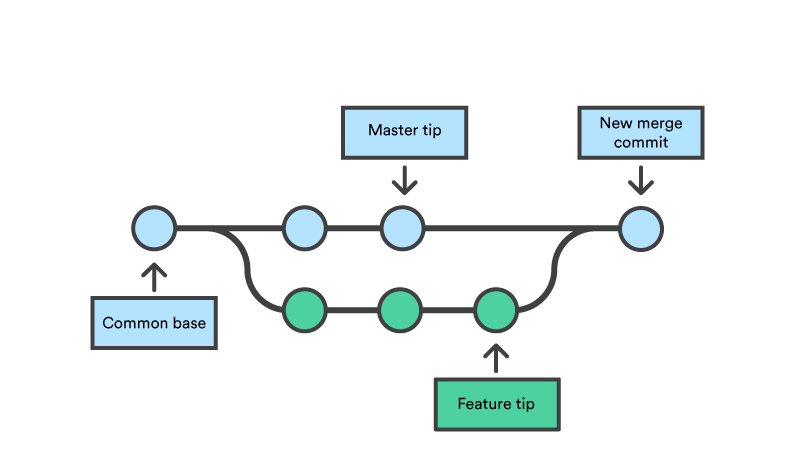
Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development.

# Git Commands

* git clone
  + This command is used to clone the remote repository from github to the local computer
* git add filename
  + This command is used to add the file to the git stage, to make the git aware of the new file. We can add multiple files by using \* wildcard
* git commit
  + This command is used to commit the files to the local git repository with the commit message; -m option is used to add a message.
* git push
  + This command is used to sync the local git repository with the remote git. It pushes all the changes done to the local files to remote git.
* git pull
  + This command is used to take the latest changes from remote git repository to local git repository.
* git checkout
  + This command is used to switch between git branches, option -b is used to create a new branch and option -D is used to delete the old branch
* git merge
  + This command is used to merge two different branches in local git repository
* git status
  + This command is used to check the status of the local repository.
* git diff
  + This command is used to find the difference between the files in local git, it is also used to find the difference between two commits
* git log
  + This command is used to find the history in GitHub

# Git Branch

 Each branch of a project saves different versions of it. Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes



By developing the features in branches, it’s not only possible to work on each one in parallel, but it also keeps the main branch free from questionable code.

# Git Stash

Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

Example:

If you are working on a branch and want to switch to a different branch and without committing the changes you can use git stash push and it will push all the changes on to a stash and make your current branch clean and the you can switch to a new branch.

If you want to pull the changes back to continue working you can use git stash pop or git stash apply.

Reference: [Git - Stashing and Cleaning (git-scm.com)](https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning#_git_stashing)

# Git Merge

Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch. This command is used by git pull to incorporate changes from another repository and can be used by hand to merge changes from one branch into another.

Assume the following history exists and the current branch is "master":

A---B---C topic

/

D---E---F---G master

Then "git merge topic" will replay the changes made on the topic branch since it diverged from master (i.e., E) until its current commit (C) on top of master, and record the result in a new commit along with the names of the two parent commits and a log message from the user describing the changes. Before the operation, ORIG\_HEAD is set to the tip of the current branch (C).

A---B---C topic

/ \

D----E----F----G----H master

# Git Rebase

Rebasing is the process of moving or combining a sequence of commits to a new base commit. rebasing is changing the base of your branch from one commit to another making it appear as if you'd created your branch from a different commit.

With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.

For this example, you would check out the experiment branch, and then rebase it onto the master branch as follows:

$ git checkout experiment

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: added staged command

This operation works by going to the common ancestor of the two branches (the one you’re on and the one you’re rebasing onto), getting the diff introduced by each commit of the branch you’re on, saving those diffs to temporary files, resetting the current branch to the same commit as the branch you are rebasing onto, and finally applying each change in turn.

Links: [git rebase | Atlassian Git Tutorial](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase)

[Git - Rebasing (git-scm.com)](https://git-scm.com/book/en/v2/Git-Branching-Rebasing)

# Git Squash

Squashing combines multiple commits into a single commit based on your commit history. With the help of squashing you can clean your branch history and can maintain an organized commit timeline. It is used before pulling requests or merging feature branches.

Links: [Git - Squash - GeeksforGeeks](https://www.geeksforgeeks.org/git-squash/)