Git Tutorial

Intro to version control

Version control is a system that records changes to a file or set of files over time so that you can recall specific **versions** later.

It is very helpful while working on any project and enables you to save it at any point and come back to it if something goes wrong. It also makes collaborating with others very easy.

A very basic method of version control is by literally making copy of the files and save it somewhere else. It works but it had certain disadvantages like it can take more space and sometime you may lose track of the backed up files or their version

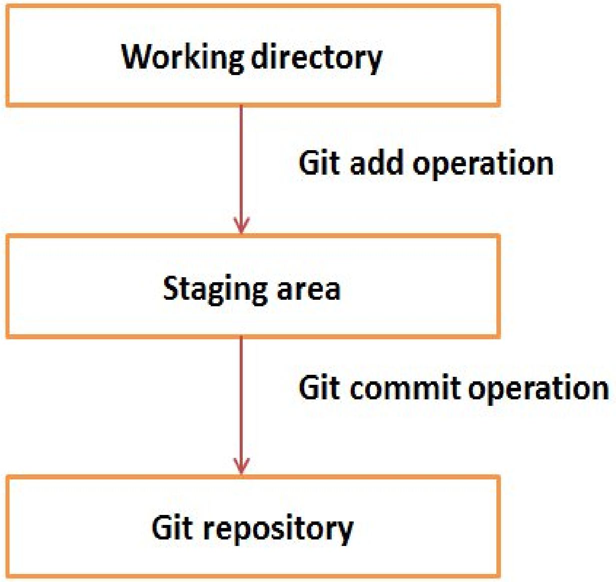
That’s where git comes in …...

How git handles version control

Git achieves something similar, it makes a copy of your files and stores in the .git folder in your directory. But git only saves the changes or the “diff” of the files from the previous version instead of the whole file when we “commit” them, this saves a lot of space.

This means that every commit is depended on the previous one. Every commit is recognised by its commit hash

Staging area and commit



Let us see the basic workflow of Git.

Step 1 − You modify a file from the working directory.

Step 2 − You add these files to the staging area.

Step 3 − You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.

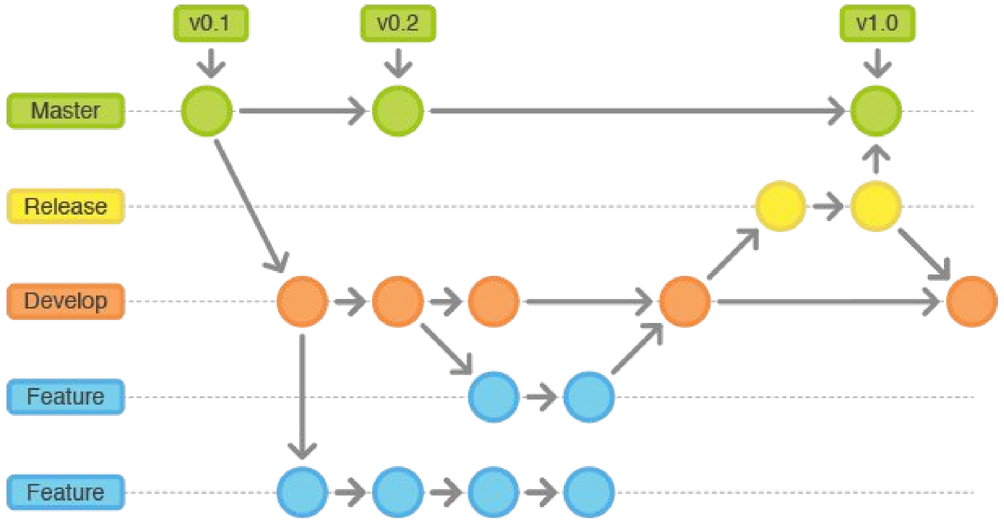
Git Stash

Suppose you are implementing a new feature for your product. Your code is in progress and suddenly some other feature request comes. Because of this, you have to keep aside your new feature work for a few hours. You cannot commit your partial code and also cannot throw away your changes. So you need some temporary space, where you can store your partial changes and later on commit it.

In Git, the stash operation takes your modified tracked files, stages changes, and saves them on a stack of unfinished changes that you can re-apply at any time.

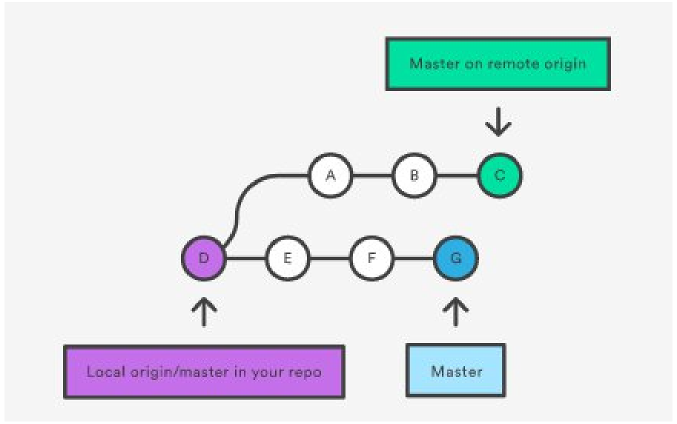
Branching

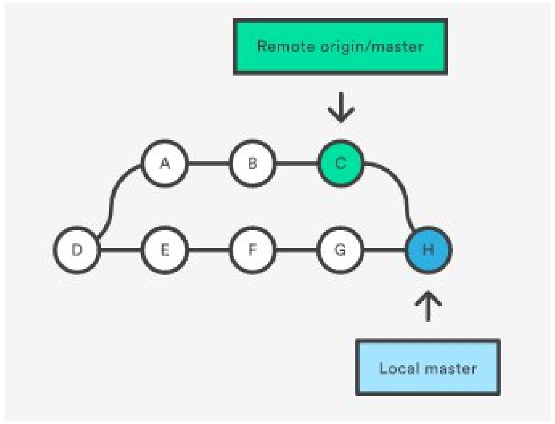
Branch operation allows creating another line of development. We can use this operation to fork off the development process into two different directions. For example, if we need a new feature to be developed along with other features, we create a branch so that the development can continue uninterrupted



Merge commit

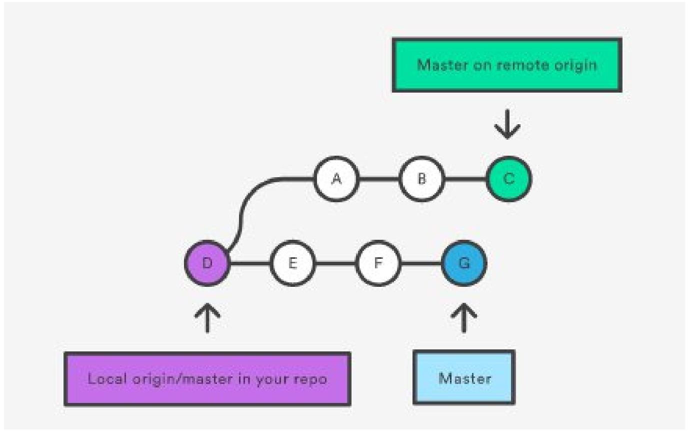
When you have 2 branches as shown below, and you want to merge the 2 branches, it does it by creating a merge commit (H in this case). This commit was not present in either branches and is a new commit

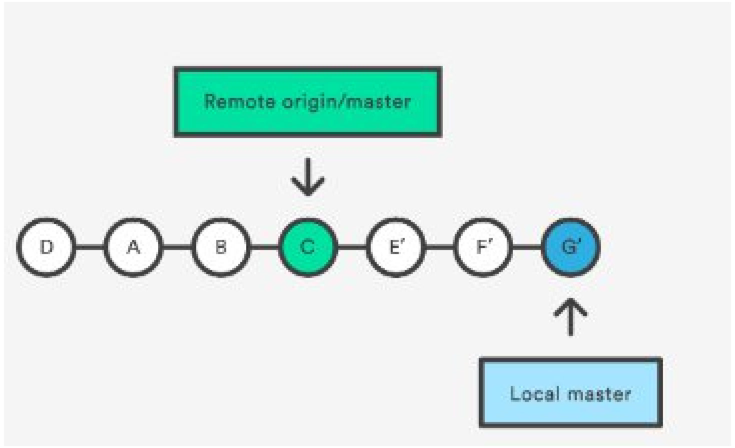




Rebase

Another way of merging 2 branches is by rebasing. While rebasing, the your current branch tracks back to the last common commit between the 2 branches (D in this case) and applies the commits from the other branch (A to C in the image) and them applies the changes in the current branch (E to G in this case). Note that no new commit was added in case of rebase and all the commits existed before the rebase operation.





Remote repository

Git is a distributed version control system. You have multiple copies of the repository in multiple devices. This include your local machine, your teammates machines and somewhere on the cloud like Github, Gitlab, Bitbucket etc.

It can act as the main hub between your work and your teammates work and also act as a backup if something happens to your device.

You can add any number of remote repositories to your project and name them as you like. Some common names are origin, upstream etc.

When you clone a repo a remote called origin is added to it which points to the repo from which you cloned it

fetch

The `git fetch` command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on.

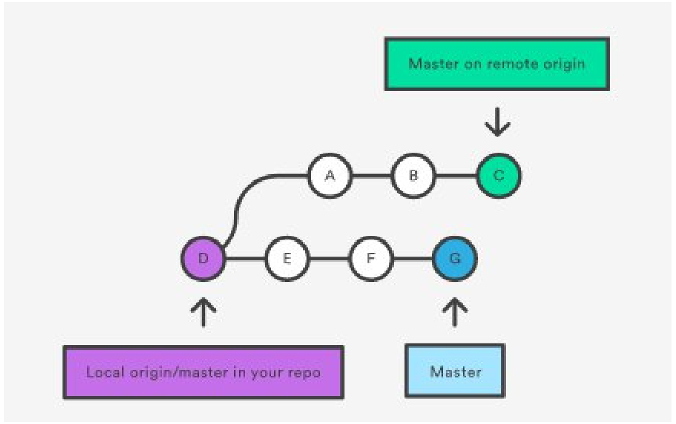
Fetch will download the remote content but not update your local repo's working state, leaving your current work intact.

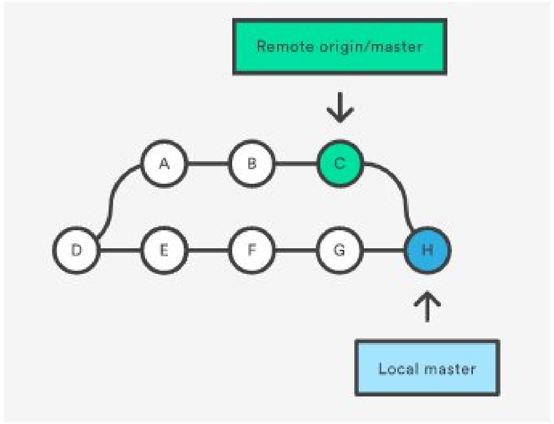
pull

Fetch and integrate changes from another (remote) repository or a local branch

Combination of 2 commands: fetch and merge

Command: git pull <repo_name>

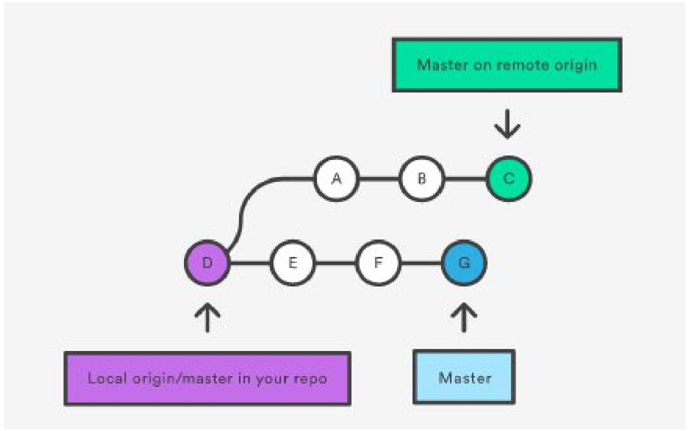


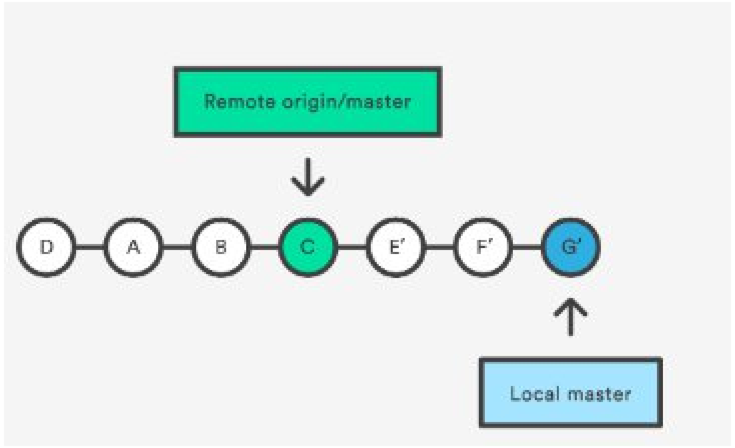


pull --rebase

Rebasing ensures that the local branch is up to date with the remote branch.

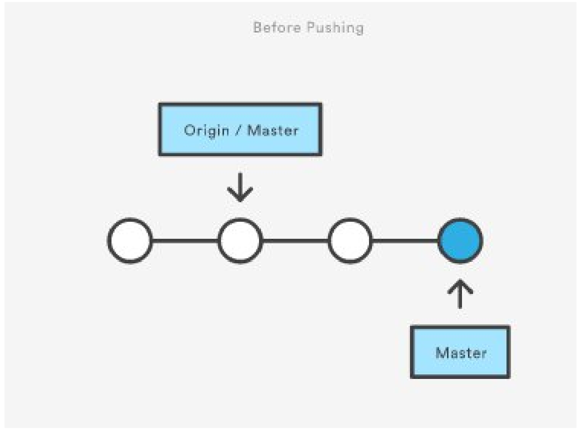
Always need to commit changes to the top of the remote branch

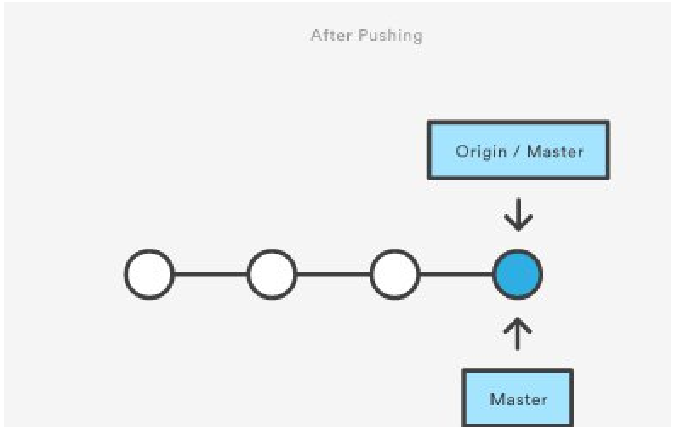




push

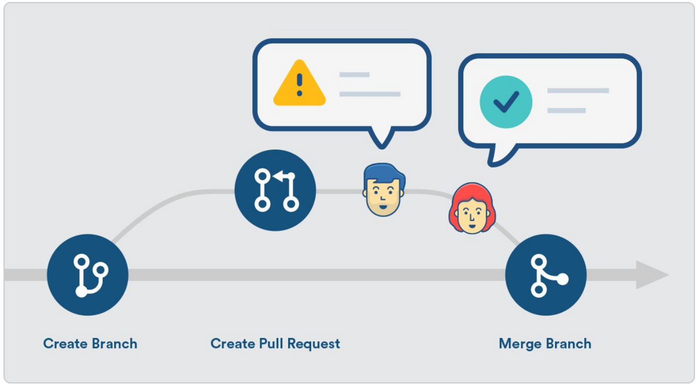
This command will push the local git repository to the remote git repository (services like Github, Gitlab, Bitbucket, etc). **Plis don’t force push.**





Pull request

Pull requests let you tell others about changes you want pushed to a branch/repository. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch/repository.



Reset

When you mess something up and want to go back to a previous commit. There are 2 types of reset: soft and hard.

Soft reset resets the commit but keeps the files intact

Hard reset resets the commit and the files. So the changes made in that commit will also be gone

Other useful commands

```
git config <setting> <command>

git log --oneline

git rm --cached <file>

git rebase -i

git cherry-pick <commitSHA>

git reflog

git clean

git squash
```

Git ignore

Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are: node_modules, .pyc, /build

Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository. There is no explicit git ignore command: instead the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore. .gitignore files contain patterns that are matched against file names in your repository to determine whether or not they should be ignored.

Normal flow

- Fork
- Clone your fork
- Add upstream remote
- Branch out
- add, commit
- pull upstream --rebase (fetch and rebase)
- resolve merge conflicts if any
- push to origin
- give pr
- If working on multiple features and if pr yet to be merged its advised to work on separate branches