Git for Beginners

(1) Git version control

Git distributed version control. System

version control → like multiple updates. (like we can go back to previous versions

Having that power of going back to any version that is called version control.

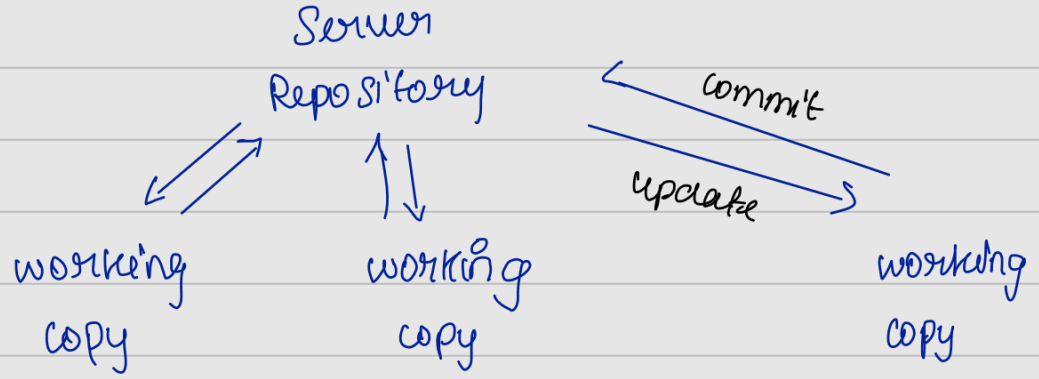
↳ System is called version control system.

Three categories

(1) local version control. → storing the content in local machine with different versions. (local copy)

* Machine failure ↳ project lost
* not collaborate with others

(2) centralized version control system (CVCS)

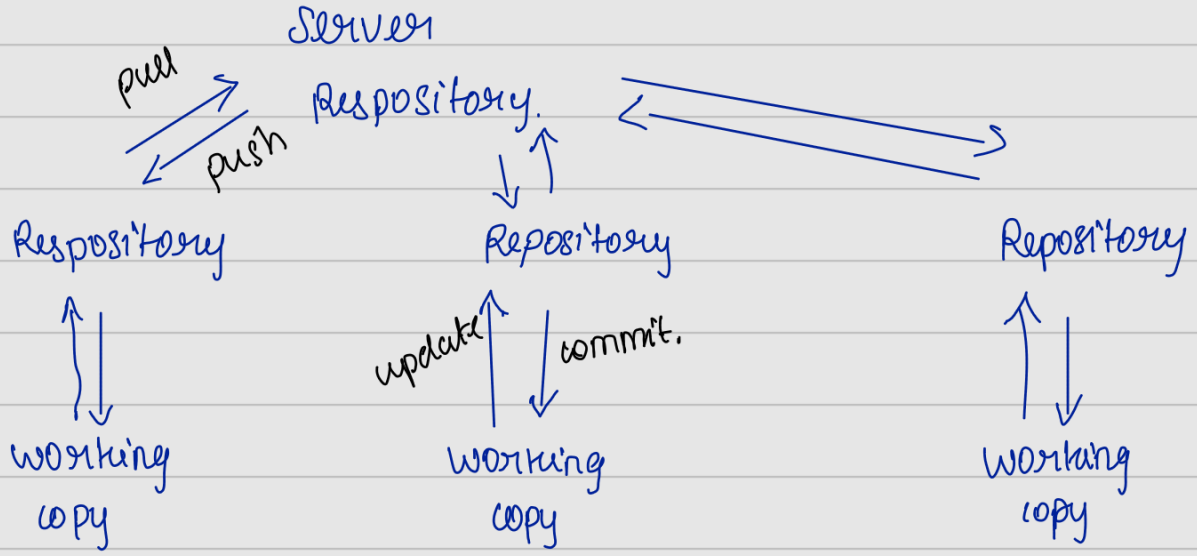


* everyone access to it.

* Everything stored on central system, if it fails game over for project.

(3.) Distributed version control system (DVCS)

* local copy + (entire commit history)
* collaborate with others as well.



Examples: GitHub, GitLab, BitBucket.

(2.) History of Git

How multiple people in the world can contribute to one project.

↳ earlier days we called something called archive files. / sending the patches.

Git is famous !

* Simple to use
* Fast
* Branching
* Fully distributed.

propriety project (software is software owned by an individual (company & is not freely available for use by the public

Linux ⇒ Bitkeeper (2002) for contribution.
(free) then Bitkeeper changed policy then Git came.

Git (2005)

③ Git Setup

Mac ⇒ xcode then Git my default.

git --version & checking for available.

when you work with GIT command, it creates 'branches'
by default you will work master branch (only one branch)
↳ now it changes main.

own identity (for collaboration)

git config
git config --global --list {username| email}

git config --global user.name "Anantha Narayanan"
git config --global user.email "....@...."

Git is for anything not only for coding

(4) Git init

Git → Not language specific

"you give me the file, you give me the statements, I will take care of it" → Git

Git should know
1. Track my progress
2. commit
3. Create different versions.

working directory

staging area
↙ Git works.
↳ place where git knows, Git wants to track.

commit history
↓
Every time you commit you get a one version of it and you can go back.

git init → local repository. (creation)
git status

By default, when you create a file, it will be part of your working directory, you have to let Git know that this should be a part of git that goes into staging area.

initialize git project → creat a Master branch.
(we need main)

git init -b main

↳ initialize the project with 'main' name and not the master.

(5) Git commit

working directory → where you edit your files (programmer)

If you want to commit something, if you want git to take care of some files for different version. you have to give that to git.

↳ add particular file to staging area.

git add . <filename> → added staging area.

git log → tell all the initial comments

This is a common saying 'commit early and commit often'

Every time you commit something you have to specify what changes you have done there.

git commit -m "First commit"

Issue tracking → solving a particular issue. (issue no)

Git also provides you something called 'integrity'. means once you save something in a git commit, you cannot change it without

git knowing about it. Git will know that something has been changed.

↳ follows checksum.
(kind of fingerprint of data)

change data → change checksum.

Git creates a checksum for every commit. Done with 'SHA-1' it creates 40 characters, but 7 characters are displayed. (in commit statement)

(6.) Git Skipping the Staging Area

So every time you make a change, add that to a staging area, then move that to the commit.

git commit -a -m "My third commit"

↳ move from working directory to commit.
(without staging area)

(7.) Git Diff

finding the differences between what you have worked and what is already there in the commit.

git diff

Now, No differences → when you added into a staging area.

so, git diff --staged.

?? → untracked.

A → added.

(8.) Git Remove file

(markdown)

.md, Readme.md → if you want to make some documents beautiful.

git add . → add all the files to staging area.

you can delete from working directory (with delete option) whereas not deleted from git.

Don't delete from git, delete from git.

git rm --cached creds.txt

(9.) Github Repository

→ Github
→ Gitlab (private features) ₹
→ Bitbucket

→ Remote Repository.

git clone → copy the entire code into your local machine.

repository

Most of the time, when your are downloading you are using HTTPS

ls -a

echo "# git-course-demo" >> README.md.

↳ echo is a command using which you can echo this message to this particular file.

HTTPS
↳ always ask for login information for pushing the information

SSH ↳ doesn't ask login information. (Once)

type README.md → to read the content of the file.

dir /a /b → to view the hidden file names.

then add SSH key to the remote respository.

(10) Adding files to remote Repository

→ Open Git-course folder
→ create a file.
→ Git status
→ git add name.txt
→ git status
→ git commit -m "user created"
→ git pull origin main
→ git push origin main.

(11) Git tag.

git push **origin** main

↳ default name of remote repository, when you clone a repository, Git automatically sets up a remote called origin that points to the repository you cloned from.

tag (version)

↳ tag is a reference that points to a specific commit in the repository, Tags are often used to make specific points in history.

git tag → show all tag

tag
- annotated tagging (release this to user, want to give more information abt user, own) ↺
- Lightweight tagging.

So, annotated tag.

→ git tag -a v1.0 -m "First release"
→ git tag
→ git show v1.0
→ git status
→ git push tag v1.0 (tag also need to pushed)

(12.) Git clone a project

git clone 'url' → download the particular repo.

git log --pretty=oneline → commit number + some information.

HTTPS → requires sign in of github account.

(13.) Git Branch create

Branch → makes Git one of the best version control system.

Git branching is a powerfull features that allows developers to create isolated environments for changes to code without affecting the main codebases.

This enables parallel development, experimentation & isolation of features / bug fixes.

To create a branch

→ checkout (old) (familiar) (-b)

→ switch (New 2.23) (-c)

main branch has no idea what you're doing with feature1 branch.

→ git checkout -b feature1

→ git branch

→ git status

→ git add name.txt

→ git commit -m '...'

→ git switch main

→ git switch feature1

(14.) Git Delete branch

→ Git switch -c feature2

→ Git branch

→ Git branch & remote repository

→ Quit switch - {move to previous branch}

→ git branch -d feature2 {delete}

→ git branch.

(15.) Git branch pushing to remote Repository

→ git status

→ git add adminService.txt

→ git commit -m 'Adding adminService'

→ git switch main

→ git switch feature1

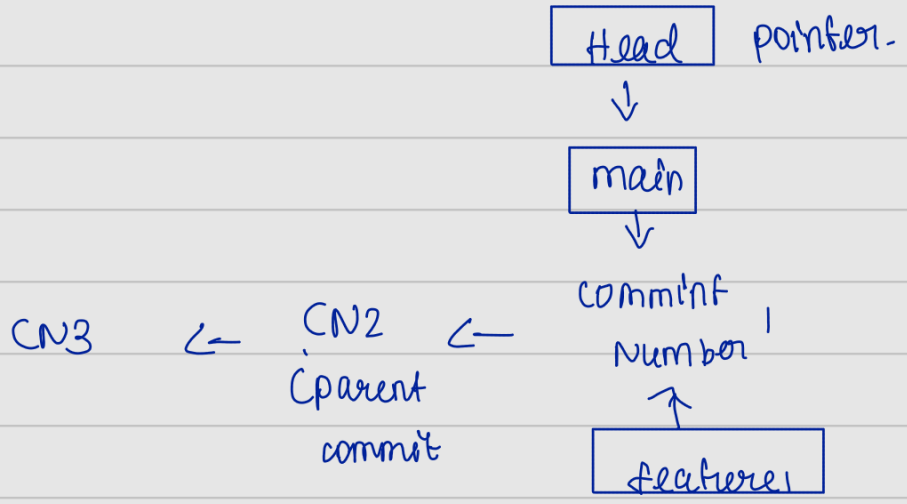
→ git push origin feature1 {push}

(16.) Git branch How it works

Git stores information about branches as references to specific commits in a directed acyclic graph.

* Each commit in Git has a unique SHA-1 hash identifier based its content
* The special reference 'Head' points to the latest commit in the currently checked out branch.
* when you switch branches using checkout/switch, Git updates the 'Head'
* 'Head' references is also updated to new commit {creating commit}
* when you merge one branch, Git creates a new commit that has multiple parent commits representing merge.

Commit → Git creates Snapshot (files) → commit number



Checksum is a hash value generated from contents of a file/objects

create a new branch, 'main' ☰ New branch is pointing to the same commit.

git log --graph {visualization}

(17.) Git Merge

Merging is a fundamental operation in version control systems, it allows you integrate changes from one branch into another.

(In main branch)

git merge feature1

(Is here update doesn't happen in the server (Github)

Always try to pull first.

git pull origin main branch.. | then, git push origin main

(18.) Git Rebase → don't see branches.

it is another way to integrate changes from one branch into another.

for clean branches & no history, just you see single line.

if you want clean tree → rebase

if you want to know what happend in each one & it happened in which branch → merge.

→ git switch -c feature3
→ git branch
→ git status
→ git add.
→ git commit -m "..."
→ git log

→ git switch main
→ git status
→ git add.
→ git commit -m "..."
→ git log
→ git rebase feature3

(19.) Git merge conflict

A merge conflict in Git occurs when the chages in the branches you're trying to merge cannot be automatically reconciled by Git. This typically happens when changes are made to the same part of a file in both branches & Git is unable to determine which changes to incorporate.

you need to manually edit the files to resolve the conflict. Decide which changes to keep / combine the changes as needed.

→ git branch
→ git switch feature3
→ git add .
→ git commit -m 'part2'
→ git branch
→ git switch main

→ git commit -m 'part2'
→ git switch -
→ git branch
→ git merge feature3
→ git add .
→ git commit -m 'part3'

(20) Git time travel

→ Most stable commit you had so that you can create a light version (may be)

→ Going back to previous commit & time travel }

* copy ID

Important thing is you have to use git checkout & the commit ID so you can go back to the previous version.

git checkout commit_hash
git checkout -b <new-branch_name> commithash
git show & view changes in commit }
git diff commit-hash1 commithash2
git revert commithash.

↳ lite-version of product.

git checkout commit_hash

↳ head get detached & not any of this particular branch now. whatever changes you make will not be affecting these branches (different zone)

21. Git stash {save without committing}

Git Stash is a convenient command that allows you to save changes that you've made in your working directory but are not ready to commit it, it's particularly usefull when you need to switch branches | perform other operation without commiting your changes.

→ git stash
→ git stash list {list of values}
→ git stash apply
↳ Saved things will be applied

22. git fork {open-source collaboration} → allowing contributors to work on changes without directly affecting original repo

fork refers to a copy of a respository that is made on an external user's account, forking is commonly used in collaborative software development, (Github)

↳ changes to be used by everyone., (not allowed) So 'fork' it

So, now clone the project make changes & you're pushing into your own repository not (Microsoft)