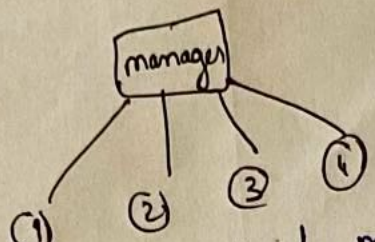
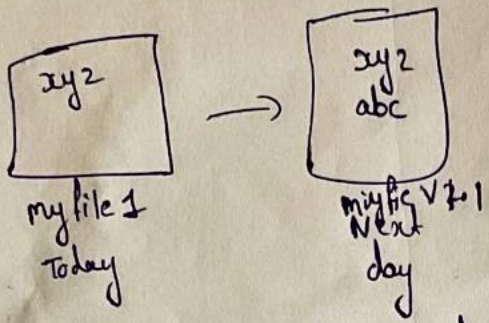
HW3

GIT

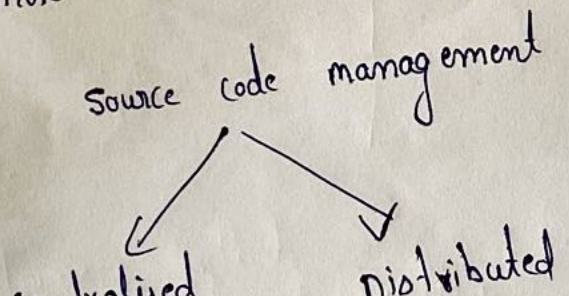
Software configuration management ⑤
source code management



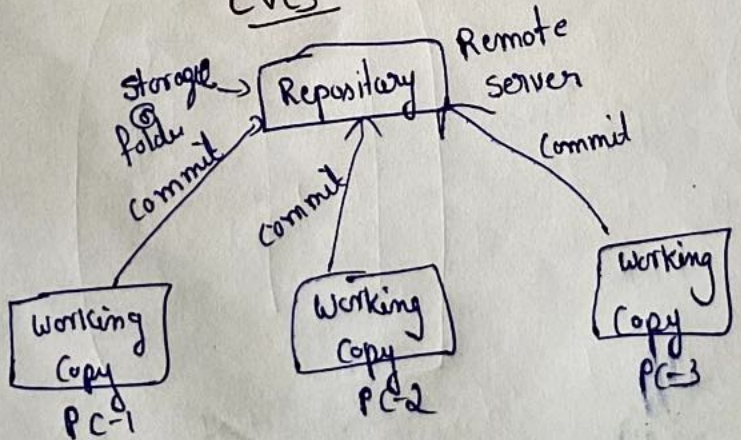
All employee send their code to manager to verify it will be tough for manager to validate.



Today we have written some code & tomorrow we will come back & add some more data so it be marked as different versions.



CVCS



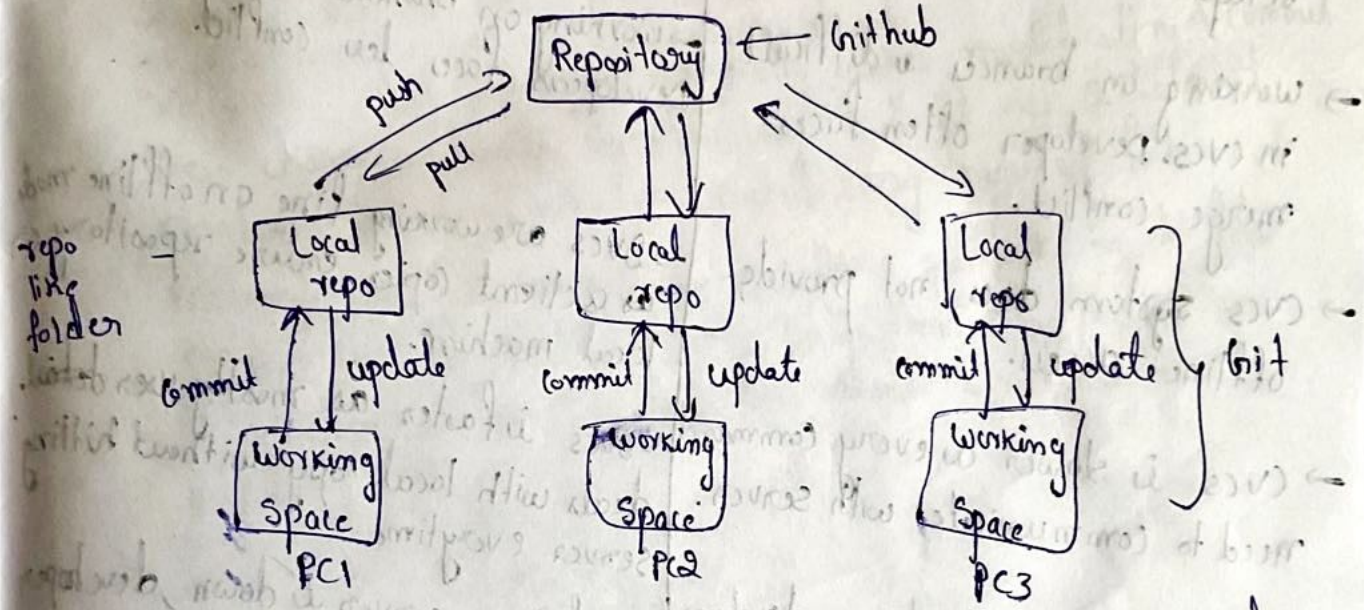
How to handle it ?

Drawback of CVCS

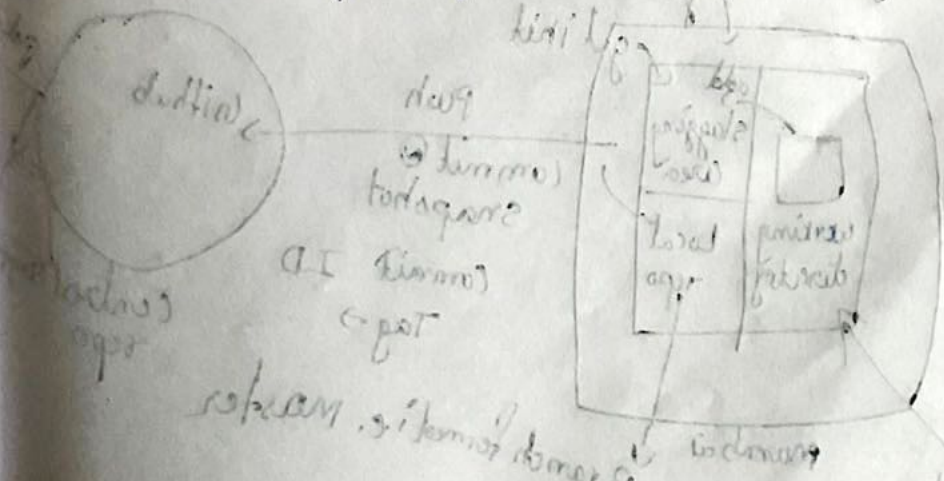
→ Remote sever get damaged means we lost data.
→ without internet we can't accer the code.

- It is not locally available meaning you always need to connected to network to perform any action.
- since everything is centralised, if central server gets failed you will loose entire data ex: SVN tool

Distributed version control system (DVCS) - GIT



In DVCS, every contributor has a local copy @ clone of the main repository i.e. everyone maintains a local repository of their own which contains all the files & metadata present in main repository.



CVCS

→ In CVCS, a client need to get local copy of source from server, do the change & commit those change to central source on server.

→ CVCS system are easy to learn setup

→ working on brances is difficult in CVCS. Developer often faces merge conflict.

→ CVCS system does not provide offline access.

→ CVCS is slower as every command need to communicate with server.

→ If CVCS server is down, developer cannot work.

DVCS (Git, mercurial)

→ In DVCS, each client can have local repo & have a comple history on it. client need to push changes to branch which will be pushed to server repository.

→ DVCS difficult for beginners. multiple commands needs to be remembered.

→ working on branches is easier in DVCS. Developers face less conflict.

→ DVCS are working fine on offline mode as a client copies entire repository on local machine.

→ DVCS is faster as mostly user deals with local copy without hitting server everytime.

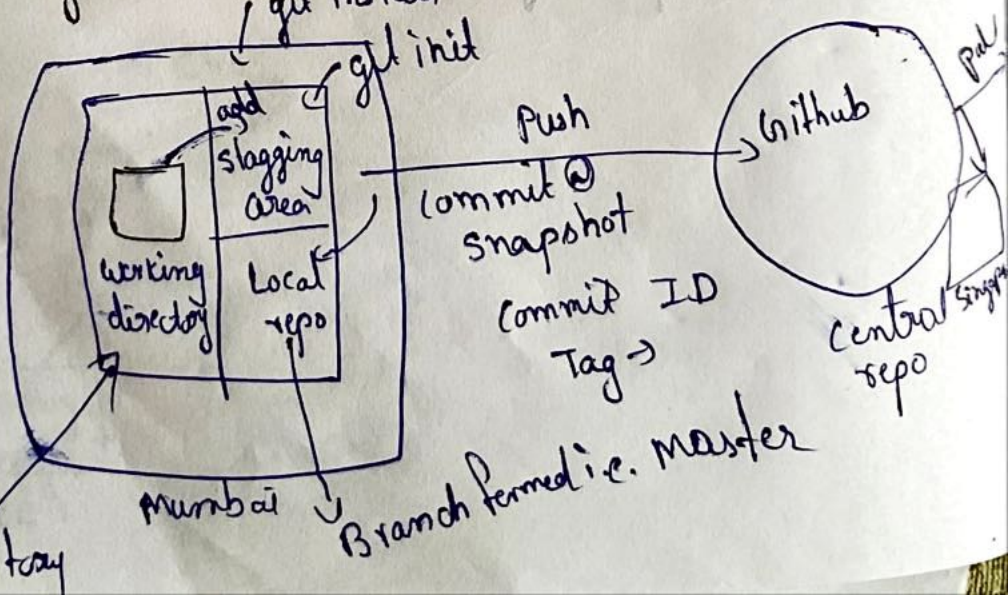
→ If DVCS server is down, developer can work on their local machine.

Git - software

Github → service

Gitlab → service

Stage of git workflow



(eg. copy the ... in ...

Repository (storage)

→ Repository is a place where we have all our code & kind of folder on server.

→ It's a kind of folder related to one product.

→ changes are personal to that particular repository.

Sever

⇒ It stores all repositories.

→ It contains metadata also

working directory

→ where you see files physically & do modification.

→ At a time you can work on particular branch.

In other CVCS, developer generally makes modification & commit their changes directly to repository. But git uses a different strategy. Git does not track each & every modified file whenever you do commit an operation. git looks for th file present in the stagging area. Only those files present in staging area are considered for commit & not all the modified file.

AWS

Commit
→ store changes in repository. you will get one commit-ID.
→ It is 40 alpha numeric characters.
→ It use SHA-1 checksum concept..
→ Even if you change one dot. commit ID will change.
→ It actually helps you to track the change.
→ commit is also named as SHA 1 hash.

Commit-ID / version-ID / version
→ Reference to identify each change.
→ To identify who changed the file.

Tags
Tag assign a meaningful name with a specific version in the repository. once a tag is created for a particular save. even if you give new commit it will not be updated.

Snapshots
→ Represents some data for a particular time
→ It is always incremental. i.e. It store the changes (appended data) only not entire copy

push
push operation copies changes from a local repository server to a remote @ central repo. This is used to store changes permanently into git repository.

pull
pull operation copies the changes from remote repository to a local machine. The pull operation is used for synchronization between two repo.

Branch
→ product is same, so one repository but different task.
→ Each task has one different branch.
→ Finally merges (code) all branches.

→ useful when you want to work parallely
→ can create one branch on the basis of another branch.
→ changes are personal to that particular branch.
→ Default branch is master.
→ File created in workspace will be visible in any of the branch workspace until you commit. once you commit, then that file belongs to that particular branch.

Advantage of git

→ Free and open source.
→ Fast and small → as most of the operations are performed locally, therefore it is fast.
→ Security → git use a common cryptographic hash function called secure hash function (SHA1) to name & identify objects within it database.
→ No need of powerful hardware.
→ Easier branching → If we create a new branch, it will copy all the code to new branch.

Types of repositories

Bare repositories (central repo)
→ store & share only
→ All central repositories are Bare repo

Non-Bare repositories (local repo)
→ where you can modify the file.
→ All local repositories are non-bare repositories

providers

git - version

Lec 14

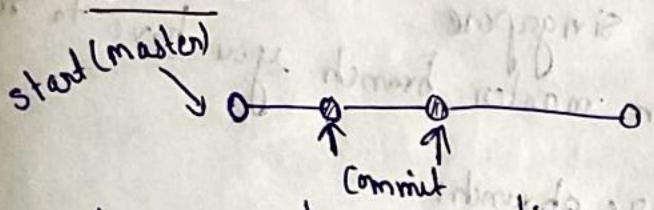
→ login to mumbai EC2 instance.
→ create one directory & go inside it.
→ git init
→ touch myfile
→ git status
→ git add .
→ git commit -m "1 commit from mumbai" (→ message)
→ git status
→ git log
→ git show <commit-id>
→ git remote add origin <central git url>
→ git push -u origin master (enter UN & PW)

Now go to singapore EC2 instance
create one directory & go inside it
→ git init
→ git remote add origin <github repo url>
→ git pull -u origin master
→ git log
→ git show <commit-ID>.
Now add some code in the file
→ git status
→ git add .
→ git commit -m "singapore update 1"
git status
git log
git push origin master

Git ignore
To ignore some files while committing
create one hidden file .gitignore & enter fileformat which you want to ignore
For eg → vi .gitignore
enter *.css
*.java

→ git add .gitignore
→ git commit -m "latest update execute .css"
→ git status
→ create some text ,java & css files & add them by running git add
For eg: touch file1.txt file2.txt file3.css file4.jav
→ ls
→ git status
→ git add.
→ git status
→ git commit -m "my text file only"

Branches



The diagram above visualizes a repository with two isolated lines of development, one for a little feature, & one for a longer running feature. By developing them in Branches, its not only possible to work on parallel but also keeps main master branch free from error.

→ Each task has one separate branch.
→ After done with code, merge other branches with master.
→ This concept is useful for parallel development.
→ You can create any no of branches.
→ Changes are particular to that branch.
→ Default branch is "master".
→ File created in workspace will be visible in any branch workspace until you commit. Once you commit then that file belongs to that particular branch.
→ once created new branch, data of existing branch is copied to new branch.

To see list of branches available
git branch

git log --oneline

Create a new branch
git branch <branchname>

git checkout <branchname>

provider

git branch -d branch1
If you want to delete that branch we use it.

Forcefully we should delete

merge

You can't merge branches of different repositories we use pulling mechanism to merge branches

git merge <branch-name>

To verify the branch git log

-> copy paste will happen in merge from one branch to another

To push to github
git push -u origin master

-> we can't merge mubai & singapore.

Ex: Suppose if you are in master branch you have to merge Branch1
git merge <branch1>

Git conflict with merge

when same file having different content in different branches if you do merge, conflict occurs (Resolve conflict then add and commit)

-> conflict occurs when you add merge branches.

Example.

master
File2
my name is vandana

Branch1
File1 ->
I am a engineer

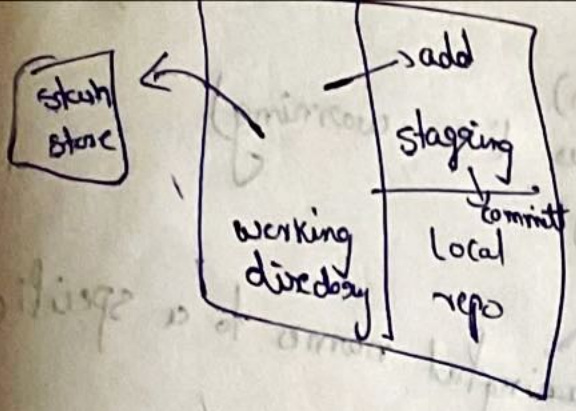
we can resolve it by add some data @ editing data in master branch file then give add & commit

Git stashing (Temporary store in local repo)

Suppose you are implementing a new feature for your product. Your code is in progress and suddenly a customer escalation occurs. Because of this, you have to keep aside your new feature for hours.

You cannot commit your partial code & also cannot throw away your changes. So you need some temporary storage. when you can store your partial change & later on commit it.

-> Stash (only applies to modified files not new files)



git stash

git stash list

To apply stashed items = git stash apply stash@{0}

Then you can add & commit

To clear stash items

git stash clear

Git reset

Git reset is a powerful command that is used to reset undo local changes to the state of a git repo

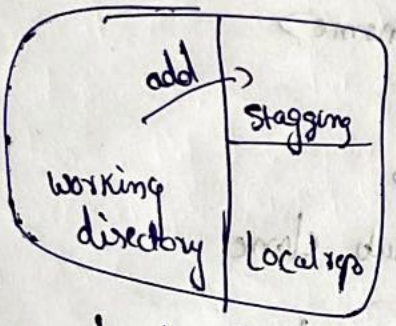
To reset stagging area

git reset <filename>

git reset .

To reset the changes from both stagging area & working directory at a time

git reset --hard



After adding to stagging area you want to delete file in stagging area

Git revert

The revert command helps you to undo existing commit

→ It does not delete any data in this process instead. Rather git creates a new commit with the included files reverted to their previous state. So your version control history move forward while the state of your file move backward.

reset → before commit

Revert → after commit

Commands: sudo su

cd mumbaigit

ls

git status

cat > newfile

ctrl+d Hi, finalcode

→ git add

→ git commit -m "code"

→ git log --oneline

→ git revert <commit-id>

How to remove untracked file

git clean -n (dry run)
(here n means it shows like warning)

git clean -f (forcefully)

Tags:
Tag operation allows giving meaningful name to a specific version in the repository.

To apply tag:

git tag -a <tag.name> -m <message> <commit-id>

Ex: git tag -a important -m "This is imp" c734a2a

To see list of tags
→ git tag

To see particular commit content by using tag
git show <tag name>

To delete a tag
git tag -d <tag name>

Github clone

→ open github website.
→ login & choose existing repository
→ Now go to your linux machine & run command
git clone <url of github repo>
It creates local repo automatically in linux machine with same name as github account.

Git cherry pick

git cherry pick enable arbitary Git commit to be picked by reference & append to current working HEAD. Cherry pick is a art of picking commit from a branch & applying it to another.

cherry-pick can cause duplicate commit so traditional merges are used instead.

git cherry-pick commithash → commit id

Components of Terraform

11. datafile, Provisioners, Backends.

AWS

Source code management tool, VCS (version control sim) → multiple version of a file
(tools used to maintain source code developed by developers) ⇒ GIT (used to push data from local to internet)

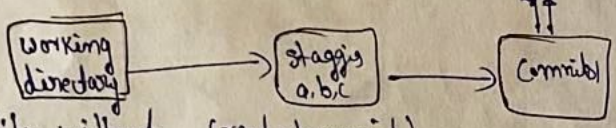
Git architecture consists of 3 stages:

working directory staging area local repository

(When you install Git, it creates these 3 folders)

working directory = new files, all existing files
staging area = whatever the files you have to move to internet that files we should move here from working directory.
(staging area is a place where Git used to take snapshots in order to maintain version)

local repo = files will get from staging area.
(commits (zip folder) ⇒ instead of saving photos differently we will create a folder then we will push to internet (like a commit, commit))



- red colors (untracked files - without authors)
(ready to commit)
VCS (version control system) → green

For example, if we have a file we have added some data & someone else comes & update the data then older data is gone. version means new modification data will be considered as new version data.

Git can be used in 2 ways

- GUI (Graphical user interface)
(whatever operations you will do done by mouse clicks)

- bash (most companies use this)
(every operation should be done by passing commands)

git init => initialise the git

git add filenames => move files from working directory to stagging area

git add . (it moves all the files from working directory to stagging if we have too files also it moves)

git commit -m "labelnames" file (123 are file names)

(git commit -m "xxxx" abc (git commit -m "file123" 123

git push => move commit to internet

git status => files to check which are available in working directorys stagging area

git log => to see commits

(Git architecture @ Git workflow will be same that diagram)

→ touch to create files

, touch 1 2 3 4 5 .. like that

(Note: only one user can be given to git not multiple)

(without configure we can't commit the data)

git config --global user.name xxxx

--global throughout laptop its going to use your name

git config --global user.email vandana@gmail.com

git config --global (it will list all the configures)

git config --global --list

→ To check what are the files in particular commit

git show ea76a710-- ((commit Id) => we should copy min 7 characters

git log -n 1 (it shows latest 1 commit)

git log -n 2 (it shows latest 2 commit)

git log --oneline

GR. b2efo50 (HEAD -> master) file5 (it shows (ommit 7 digit no & filename)

59d9d59 file4

ea76a71 file123

→ git clone URL (received from github) (URL - github - code)
once we go to that folder git will be initialized.
cd amazon

(master & main both are same)

touch - to create empty files
cd - - change directory
mkdir - - to create directory
cat - - to create & edit the file
vi - - editor to modify the files

https: (secured)
http (not secured)

(cd C:/vandana/git/) ⟹ cd dummy-file (folder name)

→ Error after uploading files to git ⟹ setting ⇒ Branch ⇒ ⇄ (change it to master)
→ Error in Git installation ⇒ windows security ⇒ App & browser control ⇒ exploit protection setting ⇒ System setting ⇒ Force randomization for randomize memory should be off

Steps

→ git remote add origin (URL) ⇒ Here we can give any nick name
→ git remote
→ git commit -m "file1"
→ git push -u origin master

{ git init
git remote add origin URL
git commit }

In setting.json file (window powershell - setting) we should add code for gitbash

git log options

git log --author vandana

-n => shows no of commits

--oneline => shows commits in single line

--author => shows specified user commits

git log --author vandana

--since

--until

git log -n 2 --author Kumar123 --oneline

→ git branch (to check which branch you are)

→ To create a branch we should use

git branch student

→ If we want to move to that branch then we should give

git checkout student

(* symbol will be showing infront of main @ master before

that is the (main) root branch)

→ To delete the branch

git branch -d student

→ Before deleting student branch we should switch to main branch

git checkout main

git branch -d student

→ Instead of creating branch & switching to that branch we can use

git checkout -b vandana

→ merge } Copy the commits from one branch to another

rebase }

→ (whichever is the main branch that will be moved to production)

→ To move vandana branch as main branch

First go to main branch then enter following code

git merge vandana

providers

https => Username & PW

SSH => public key & private key

ssh ssh-keygen (then enter 3 times)

go to that path using cd /c/Users/rakesh/.ssh

give ls

you can see the files as id_rsa id_rsa.pub

→ public key in our repository & private key with us

cat id_rsa.pub.

→ copy the content & paste in Github account repositing.

setting → SSH & GPG keys → Give title as you wish such as key & description that you have copied when you displayed cat command

To check SSH is working properly @: not.

→ Delete all the files which are in that folder locat.

→ go to that folder path.

→ git clone SSH path from Github account

→ ls (you will get all the files here

merge :- fast fwd => 1 2 3 4 5 6, It moves 6 to main branch

-- recursive = main = 1 2 3 4 5 6 = 1 2 3 4 5 6 7 (main)

rebase student = 1 2 3 6 5 7

(By default it takes fast fwd (1) recursive (2) no need to choose ourself)

-> If both the branches have changes we call it as recursive only one means main fast forward.

-> we should do merging from main branch @ master branch

(git checkout 'master' -> whichever we want to move)

-> If recursive merge happens that extra commit will be added but it doesn't contain any file it's just showing reference

Ex. merge branch 'student'

rebase => it will take common commits & prints source commits first then master commit but vice-versa in merge.

Ex: random = a, b, commands, 7, 1, 8, 9, 11

master = ~~3, 7, commands, b, a~~

= a, b, commands, 3, 7, 12, 13

o/p: ~~a, b, commands, 7, 3, 7, 1, 8~~

o/p: a, b, commands, 7, 1, 8, 9, 11, 3, 12, 13

merge

-> extra commit in recursive

-> In real time we use merge because in rebase commit order is changing

rebase

-> no extra commits

(project is linear)

→ In git we should not delete any commit

→ If you want to move any commit from repository to stagginy area we should we reset

git reset --soft commit Id (But we should give the below commit Id not the one which we want to move)

(whatever commit we have on top that will be moved to stagginy area)

reset - 3 option

--soft (filed moved to stagginy area)

--mixed (files moved to working directory)

--hard (we won't we)

→ we can do modifications of files in working directory not in staging

If we By now we have moved our files to stagginy area from working directory

git reset head

→ If we want to delete 2 commits.

git rebase -i head~2 (here 2 means 2 @ 1)

(In visual studio it shows pick we need to change it to drop)

→ If we want to rename 1 commit

git rebase -i head~1

(We should give reword @ to change name of the commit

⇒ one more time editor open the change the commit name)

(git amend used to rename top commit only for top)

⇒ squash used to combine the files

pick 6efd37c fileb

squash 71521a2 filelist

squash ⇒ remaing commit & adding files to previous commit, commit messages

(To perform merge & rebase operation working directory & staging area should be clean)

→ stash memory will be in between staging area & local repository (it's optional one)

For Example 11 12 13 in staging area

git stash save vandana

If we give git status 11, 12 13 files won't show in staging area

stash

create => stash - git stash save vandana

list => . git stash list

git stash pop

git stash apply

git stash drop (to delete stash)

To check file is existed in stash or not,

git show stash@{0}

To remove file from stash

git pop stash@{0}

If we want to move one more file to same stash give same name. git stash save vandana

* latest files which are coming moved into stash {0}

apply => copy-paste git stash show stash@{1}

git stash drop stash@{1}

what are git commands that you use to commit changes to your remote repository

git add
git commit
git show

git fetch

Downloads changes from remote repository but does not modify your local branch.

git fetch origin

git pull

Downloads changes & automatically merges them into current branch.

git pull origin main

git merge

→ creates non-linear (not sequentially) commit history with merge commit.
→ creates a new merge commit.
→ Typically used when you want to preserve the full history of changes.
→ safe for shared branches as it doesn't rewrite history.

git rebase

creates a linear commit history by applying commits on top of target branch.
→ Does not create a merge commit.
→ Typically used when you want a clean, linear history often before merging a feature branch into main/master
→ Risk for shared branches because it rewrites history.

what is .git & .gitignore

.git → Contains all the information needed by Git to track your repository (history, branches, configuration)

→ Hidden folder located at root of the repository. (eg: .git/)

→ .git is not versioned & should never be committed.

.gitignore →

Specifies which files & directories Git should ignore & not track in repository.

→ File located at root of the repository

→ Yes .gitignore is versioned & shared across the repository So all the collaborators will have same ignore rules.

What are pre-commit hooks & post-commit hooks?

Git hooks are scripts that run at various points in the Git workflow to customize & extend's Git functionality.

pre-commit:

→ Run before commit

→ If the hook script returns a non-zero exit code the commit is aborted.

→ If script succeeds the commit proceeds as usual.

post-commit hooks:

→ Runs after commit is created

→ No impact on commit itself

Webhook:

→ web hook are used to send real-time data from one system to another when an event occurs.

→ They are event driven with receiving systems having a URL endpoint that listens for incoming data.

→ webhooks are useful for integrating services & automating workflows.

→ webhook are typically configured by providing an endpoint URL & selecting specific event that will trigger webhook.

component

provider

AWS

git clone

git fork

branching strategy → main branch, developer branch, feature branch

Git branching strategy

long lived → master, development

short lived → feature, release, hotfix

PROD

new feature