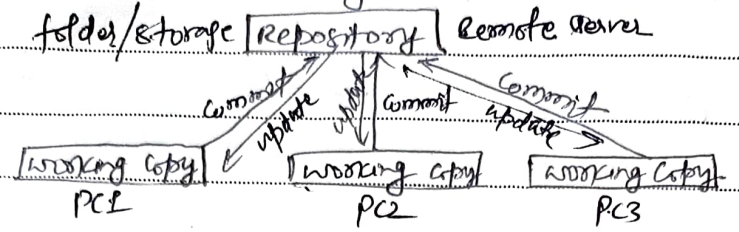
GIT

Software Configuration Management / Source code Management

Centralised Version Control system (CVCS)

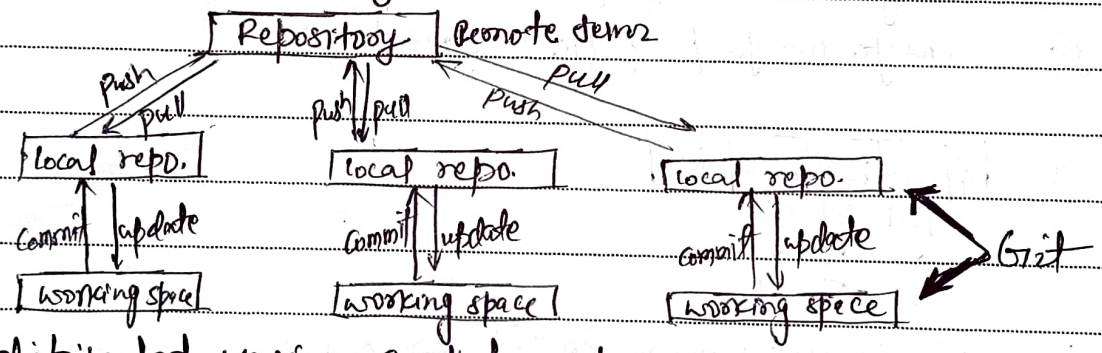


• Drawbacks

→ It is not locally available, meaning you always need to be connected to a network to perform any action.

→ Since everything is centralised, If central server gets failed, you will lose entire data. Eg:- SVN tool

Distributed version control system



→ In distributed version control system, every contributor has a local copy or clone of the main repository i.e., everyone maintains a local repository of their own which contains all the files and metadata present in the main directory.

Advantages of Git

→ free and open source, it's fast & small because most of operations are performed locally.

→ No need of powerful hardware.

→ Easier branching because if we create new branch it copy all the code to new branch.

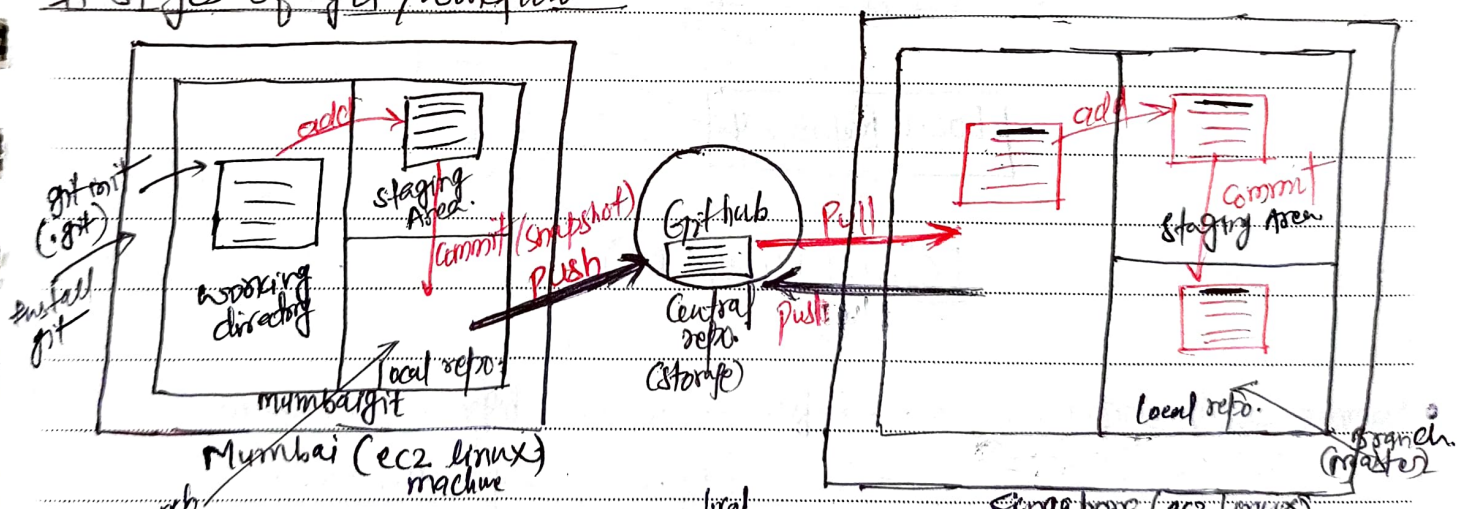
→ Security → uses common cryptographic hash function (SHA1) to name and identify objects within its database.

- Bare Repositories (central repo)
  → store and share only
  → All central repo. are bare repo.

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

CVCS	DVCS
→ In CVCS, a client need to get local copy of source from server, do the changes and commit those changes to central source on server	→ In DVCS, each client can have local (repo) branch as well as a complete history on it. Client need to push the changes to branch which will then be pushed to server repository.
→ CVCS system are easy to learn and setup	→ DVCS are difficult for beginners, multiple commands.
→ working on branches is difficult in CVCS, developer often face merge conflict	→ working in branches is easier for developers and faces less conflict
→ CVCS system donot provide offline access.	→ DVCS system are working fine offline mode (local machine)
→ CVCS is slower as every command need to communicate with server	→ DVCS is faster as mostly user deals with local copy without hitting server every time
→ If CVCS server is down developers cannot work	→ If DVCS server is down developer can work using their local copies.

Stages of git/workflow



git init → directory convert to local repository

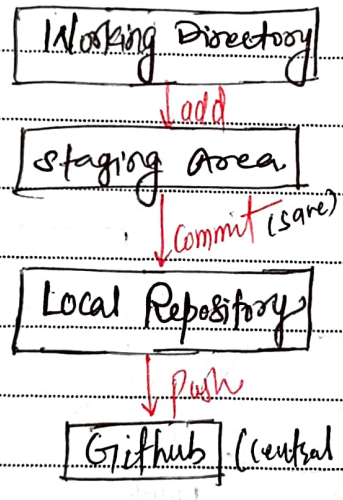
- workspace/working directory
- staging Area
- local repo

- staging area से जिसको भेजना है उसको add करते हैं।
- Everytime new commit has new commit id.

Terminology

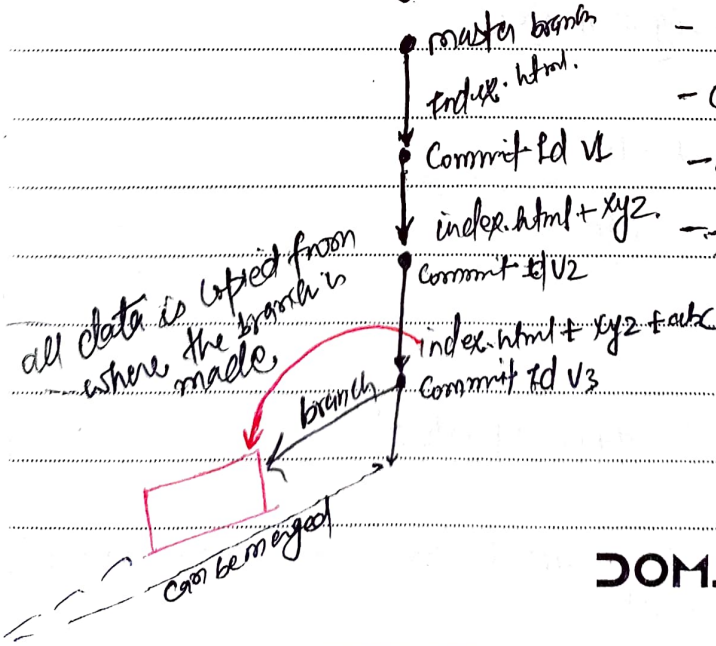
- Repository - Place where you have all your codes or kind of folder on screen
  - Kind of folder related to one product.
  - Changes are personal to that particular repository.
- Server - It stores all local repositories
  - It contains metadata also
- Working Directory/Space - where you see files physically and do modification
  - At a time, you can work on particular branch.
- In other CVCS, developers generally makes modifications and commit their changes directly to the repository. But git uses a different strategy, Git doesn't track each and every modified file.

whenever you do commit an operation, git looks for the files present in the staging area are considered for commit and not all the modified files.



- Commit-ID/version-ID/version - Reference to identify each change
  - To identify who changed the file
- Tags - Tags assign a meaningful name with a specific version in the repository. once a tag is created for a particular date, even if you create a new commit it will not be updated.

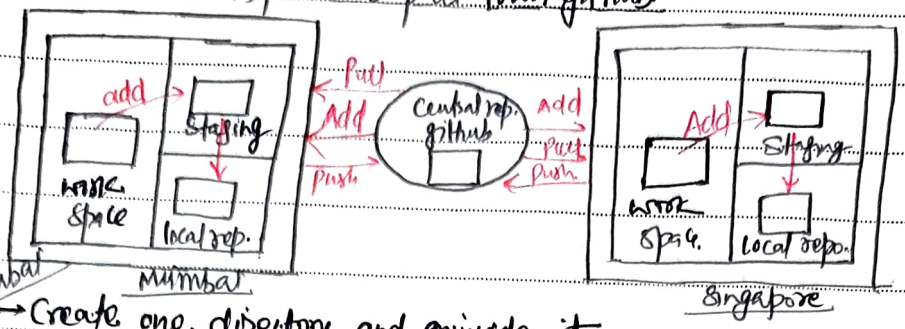
- Snapshots – represents some data of particular time.
  - It is always incremental i.e., it stores the changes (appended data) (increase) only, not entire copy.
- Commit – store changes in local repository. you will get one commit id.
  - It is 40 alpha-numeric characters.
  - It uses SHA-1 checksum concept
  - even if you change one dot, commit ID will get changed.
  - It actually helps you to track the changes.
  - Commit is also known as SHA1 hash
- Push – push operations copies changes from a local repository ~~instances~~ server to a remote or central repo. This is used to store the changes permanently into the git repository
- Pull – pull operation copies the changes from a remote repository to a local machine. the pull operation is used for synchronisation for two repo.
- Branch – Product is same, do one repository but different task.
  - Each tas has one separate branch.
  - finally merge (code) all branches.
  - useful when you want to work ||ly.



- Can create one branch on the basics of other
- Changes are ~~particular~~ 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, than that file belongs to that particular branch.

• for clearing terminal → control + L

How to commit, push and pull from github



→ Create one directory and go inside it.

→ git init (repository बना दी directory को) (.git → repo)

→ touch / cat filename {Put some data}

→ git status

→ git add . (added to staging)

} to add anything new repeat from here. आगे का steps करने है।

→ git commit -m "1st commit from Anshu" ("" represents message)

→ git status (Now this will be empty as it was added to local repo)

→ git log (कितने commit किया) (यहीं पे commit id दिखेगा)

→ git show <commit-id> (show करेगा जो code लिखा गया है।) (commit id के starting kuch)

→ git remote add origin <centralgit पता> (Add to github)

→ git push -u origin master

(enter username and password)

Go to Singapore etc → create directory

→ git init

→ git remote add origin <github rep पता>

→ make directory

→ git clone <पता of repo>

→ git pull -u origin master

→ git log

→ create branch and do changes

→ git show <commit id>

→ git diff

Now add some code in file

→ go to default branch (git pull) to check new updates

→ git status

→ git add .

→ git merge (git pull)

→ add, commit, push

→ git commit -m "Singapore update 1"

→ git status

→ git log (यहाँ पर आपने जो commit किया दोनों दिखेगा)

→ git push origin master

→ add → staging Area में

Ignore

→ To ignore some files while commiting (मान लो git में अगर ec2 user के लिए)

→ Create one hidden file .gitignore and enter file format which you want ignored

Eg:- Vi .gitignore (किसी dir में जाके)

.css (mean, file1.css ⇒ file1 ~~*~~)

*.java

→ git add .gitignore (staging Area)

→ git commit -m "latest update excludes ess"

→ git status

→ Create some text, java and css files and add them by running "git add"

Eg:- touch file1.txt, file2.txt, file3.java, file4.css

→ ls

→ git status (ignored file नजर नहीं आएंगे।)

→ git add .

→ git status (सिर्फ unignored नजर आएंगी।

→ git commit -m "my textfiles only"

→ git log

→ : ⇒ q press करके बाहर आजाएं।

→ git show <commit id>

Tags

→ Tag operations allow giving meaningful names to a specific version in the repository.

→ To Apply tag

git tag -a <tagname> -m "<message>" <commit id>

→ To see list of tags

git tag

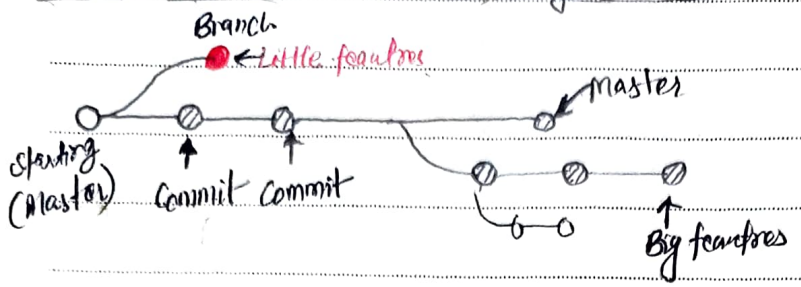
→ To see particular commit content by using tag

git show <tagname>

→ To delete a tag git tag -d <tagname>

→ ★ represents PWB

How to create Branch, merge & stash



→ The diagram above visualizes a repository with two isolated lines of development, one for a little features and one for a longer-running features. By developing them a branches, it's not only possible to work on both of them in parallel, but it also keeps the 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 personal to that branch.

→ Default branch is 'master'.

→ files created in workspace will be visible in any of the branch workspace until you commit. once you commit, then that files belong to that particular branch.

→ when created new branch, data of existing branch is copied to new branch

Some Commands

→ <directory>

git log --oneline (to see commit)

→ git branch (to show current no. of branch)

→ git branch branch1 (to make new branch) (will get copied automatically from master branch)

* master
branch1

→ git checkout branch1

→ git branch
master
* branch1

→ git add (Commit (-D for forcefully delete)
→ git branch -d branch1 (to delete a branch)

Merge

→ It Copies
→ you can't merge branches of different repositories
→ we use pulling mechanism to merge branches (जहा से pull किए वहा की रहेगा।)
→ * Master
branch1

git merge <branch name to be merged>

→ To verify the merge
git log --oneline
→ To push to central repository like github
git push origin master

कोई नया branch बनाने के बाद अगर उसी की file बनाते है तो वह दोनो branch i.e, master and branch created में नजर आएगा जब तक उसको commit नही कर देते।

Git Conflict

→ when same file having different content in different branch, If you do merge, conflict occurs (resolve conflict then add and commit)
→ Conflict occurs when you add branches
→ क्योंकि git को समझ नहीं आएगा उसको पहले लिखना है किसको बाद में।
→ Master branch में जाके vi editor से कुछ changes करके conflict को resolve करते है।
→ उसके बाद editing के बाद फिरसे merge करने की जरूरत नही है। simply add the file in staging area and commit

Git stashing.

→ Suppose you are implementing a new feature for your product, your code is in progress and suddenly a customer escalation comes because of this, you have to keep aside your new feature work for few hours, you cannot commit your partial code and also cannot throw away your changes, so you need some temporary storage, where you can store your partial changes and later on commit it.

→ To stash an item (only applies to modified files not new files)

→ To stash an item

git stash (करने के बाद empty हो जाता)

→ To see stashed items list

git stash list

{ पहले एक तो touch filename
git add.
git commit -m " "
(उसके बाद आगे काम करके फिर stash कर सकते हैं)
fixes }

→ To apply stashed item

git stash apply stash@{0} {0, 1, 2 is numbering; 0 is latest}

→ Then you can add and commit

→ stash में भी list आता है, to clear stash items

git stash clear

Git Reset

→ git reset is a powerful command that is used to undo local changes to the state of a git repo. (staging area में add करने के बाद अगर changes करते हैं तो)

→ To reset staging area

git reset <filename>

git reset .

→ To reset the changes from both staging area and working directory at a time

git reset --hard

→ reset is done on private file but revert is done on public file as its committed.

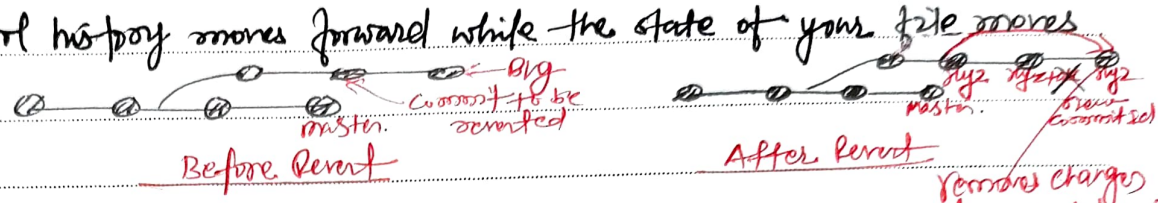
→ Commit cannot be undone

__/__/____

Git Revert

→ Revert command help us to undo an existing commit.

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



→ reset – before commit
revert – after commit

lab

→ cd dir, ls, git status

→ cat >file, Hi, final code for app

→ git add .

→ git commit -m "code"

→ git log --oneline

→ git revert <commit id> (Please ignore previous commit)

→ To remove untracked files

git clean -n (dry run)
git clean -f (forcefully)

→ Github clone

Choose existing repo on github and run (it creates a local repo automatically in machine with the same name as on github)
Git Clone <url of github repo>

→ like as git there is also master branch on github and can also be made new branches but only difference is git is CLI and github is GUI.