Design of Git.

① Snapshots

Old systems:- (CVS, SVN)

⇒Store only the differences (deltas) between file versions.

Git

⇒It takes snapshot of every commit of project

If there is no changes, git points to the previous version itself of that unchanged file.

Snapshots - photo or picture of all files present at that moment.

② Content Addressable Storage:-

⇒git stores everything (files, commits) in key-value db.

key → SHA-1 hash value:- the actual data.

⇒handles with storage

i) Mechanism:-

Data ⇒ value hash of data → key (address).

i) Retrieval of Data can be done using that address.

ii) Two identical data (files) will give the same hash. Git stores one copy, so it saves space.

iii) If a file's content changes ⇒ generates a new hash → new blob is created.

blob ⇒ Stores binary content of that files.

↳ core data units in Git.

③ DAG:-

⇒stores the history of commits as a graph. not as a simple list.

i) A new commit always points to the old commit (parent) (child). child → parent.

ii) Acyclic:- No loops are allowed, a child cannot become its own ancestor.

Why DAG:-

Merging is possible ~~when~~ as Git understand where the branches are diverged.

It finds the Lowest common Ancestor.

As it acts as the.

→ last common point before the branches are ~~updated~~ separated.

→ Helps to calculate merge results.

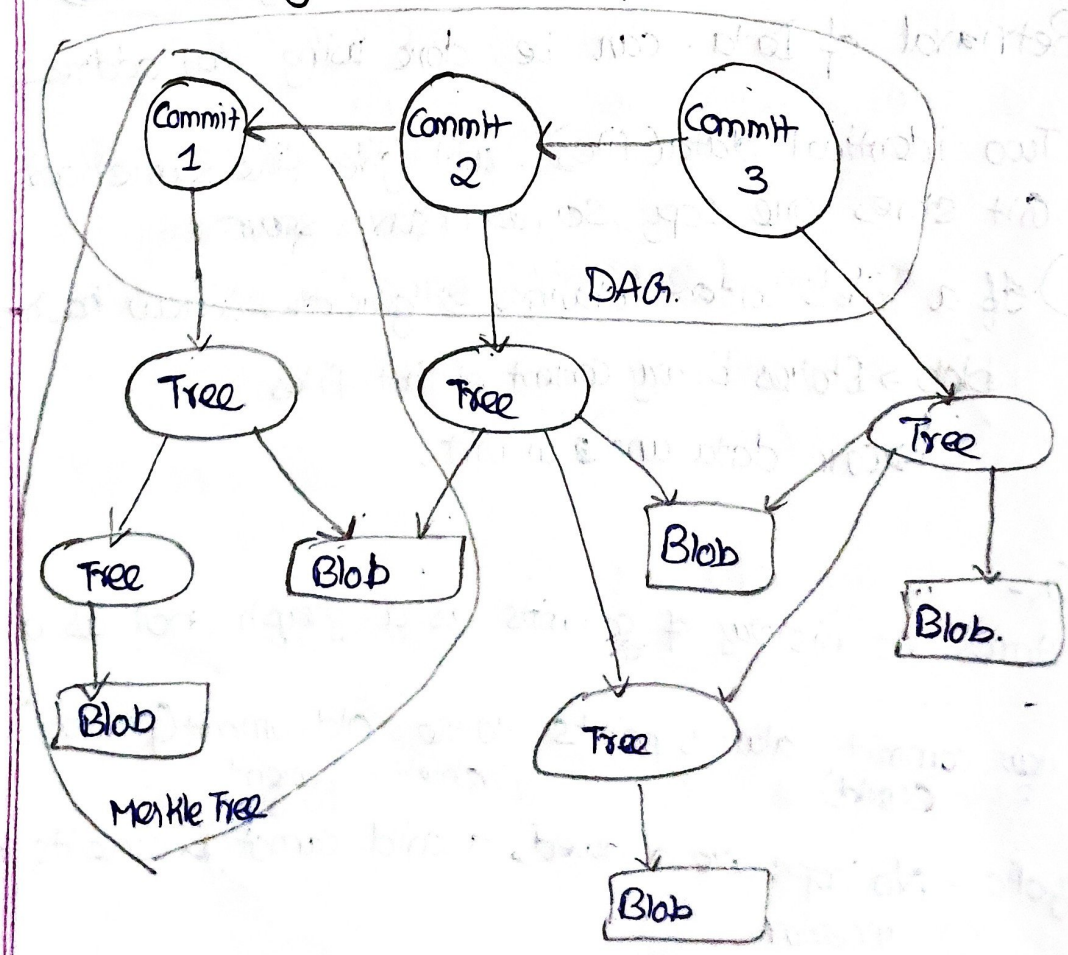
Cryptographic chain:-

→ Every commit stores its own data and hash of its parent commit.

→ A merkle Tree is created.

→ So changing one commit changes all future commit hashes.

→ History becomes tamper proof.



Decentralization:-

→ Every clone is full working copy of repo as in github.
→ No master or main copy.

i) Autonomy
   → commit, branch, view history without internet.

ii) Redundancy.
   → full backup of entire project history.

Integrity.
   → Everything stored in Git is verified using checksums (SHA-1)
   → Things cannot be changed.
      → File content.
      → History.
      → accidentally losing data.

AS
   → Commit hash depends on all data before it.
   → Changing anything changes the entire chain.

Three States:-

1) Working Directory:-
   ↳ actual files that are seen and edited.

Staging Area:- (Index):- .git/index.
   ↳ Binary file that caches metadata about working directory and lists what will go into the next commit.
   → Select which changes go into the next commit.
   → Review changes before saving them permanently.
   → git add <files>
      moves changes from working directory → staging area.

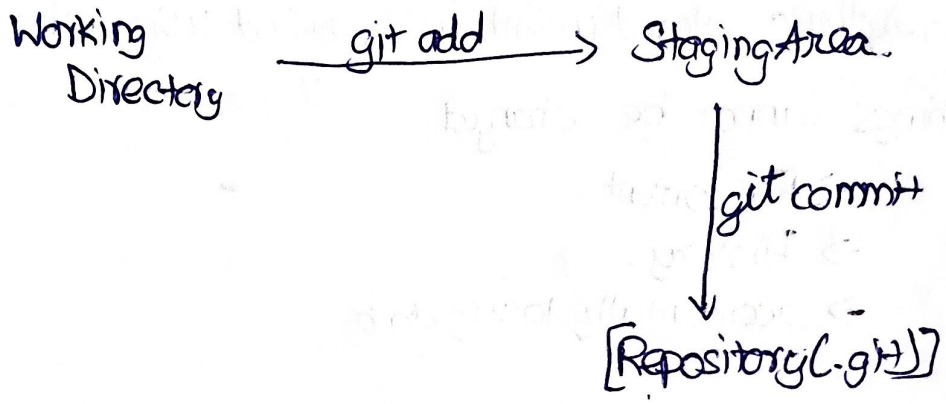
Repository:-

⇒database where git stores all commits, branches, tags and objects.

⇒ git commit.

⇒ Staged change are saved into the repo permanently.

Flow



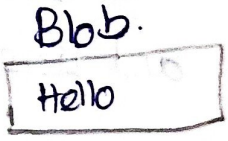
Four Object Types.

Objects are stored in .git/objects folder. compressed with zlib.

i) Blob:- Binary Large object.

⇒Stores only file's content.

⇒Not stores ⇒filename foldername, Timestamp



⇒Stores raw text.

Because:-

Git Tracks filenames through Tree objects not Blob

2. Tree object:-

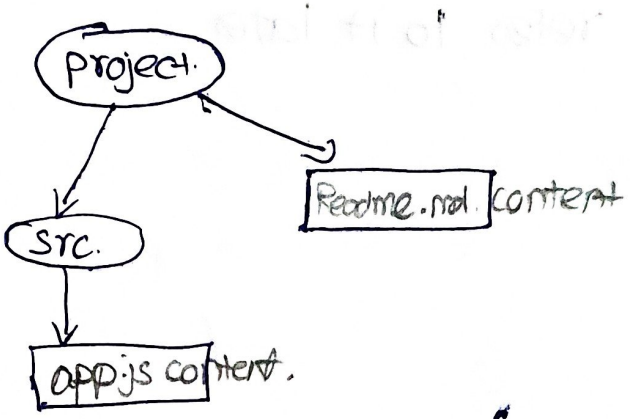
Stores :- a directory (folder).
- filenames.

Links filenames → BlobID's

Links subdirectories → other Tree ID's.

Describes:-
→ Folder structure
→ Which file exist.
→ Which blob belongs to which filename

```
project/
    Readme.md
    src/
        app.js
```



So git can rebuild entire folder structure from scratch.

3. Commit Object:-

→ wrapper object.

Contains.
i) pointer to a toplevel Tree → represent entire project snapshot.
ii) Author info. → the one who wrotes the commit
iii) committer info → the one who added the commit.
iv) Timestamp. → when the commit was created
v) Parent commit(s).
- one parent for normal commits.
- two parents for merge commits.
- no parent for first commits.

parent pointer creates the history chain forming DAG.

Tag object:-

=> human readable permanent pointer to a commit.

Examples:-

v1.0

v2.0-beta, release - 2025.

=> give a name to commit so developers can refer to it later.