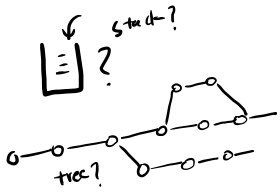
Git Advanced - How git internally works - Notes by Milind Mishra

→ To implement something like git, what will you require?
How does git work behind the scenes... ✓



git is heavily dependent on hashing, graph/tree data structure

git → overall like a key, value store.

key: value

Stores hash of data

Stores data.

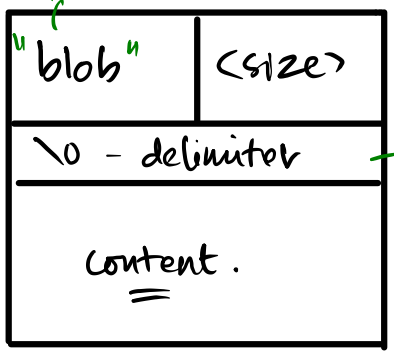
↓ {Secure hashing Algo} (fn)

git uses a cryptographic hash fn SHA-1, for a given data → o/p 40 digit hexadec. no.
the hash is always same for same kind of data.

* git compresses the data & stores in a blob & stores some metadata about the data stored.

~ all these files are stored @ (.git) folder ✓

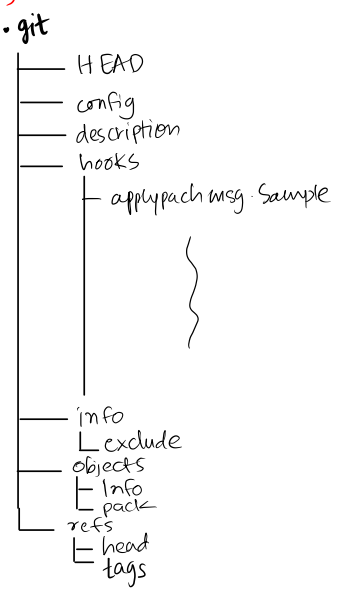
Metadata



* tree (.git)

folder structure of .git folder:-

{.git} folder structure :-



8 directories, 17 files.

$ echo 'Hello git' | git hash-object --stdin

input — Pipe operator — cmd

generates (same hash) — for the cmd.

0dec2239efc0b1fabe4078f

NOTE:- Same hash for same data!

write inside git repo:

$ echo 'Hellogit' | git hash-object -w --stdin

{ makes object in object folder of .git dir. }

thats how git stores data inside objects/ folder.

To visualize .git folder: (tree .git)

ex: $ git add README.md → hashes it & stores at objects/.

adds: objects / 0d / hash of file. ~

first 2 chars of hash

objects / 78 / remaining hash

$ git cat-file -t objectref

-p

$ git commit -m "added README"

blob: binary large object (type)

$ cat object ← spits contents of file

↳ spits blob → gibberish.

40 char HASH

object/ ↓

first 2 char

└ 38 digs. of. hash

view using cmd

$ git cat-file -p <hash>.

prints file-contents of blob. — flag.

16 16 16 16

40 digs.

* Problem of hashing. – collisions → partial or full. ⟶ solve in "probing".

– 2 files inside same folder ✓

$2^{160} - 10^{30}$ Exabytes → data can be stored.

* all code changes reside at object folder w/ their respective #'s.

40 dig. hexadecimal ✓ (hugo)

* tree – it contains pointers (using the hashes)

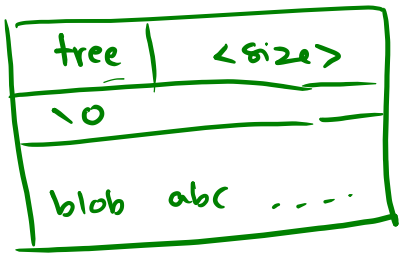
↓ → to store blobs.

directed → to store trees.

and metadata ___ type of pointer (blob)

└ directory file name

└ mode (R, W, Ex. .. etc.)



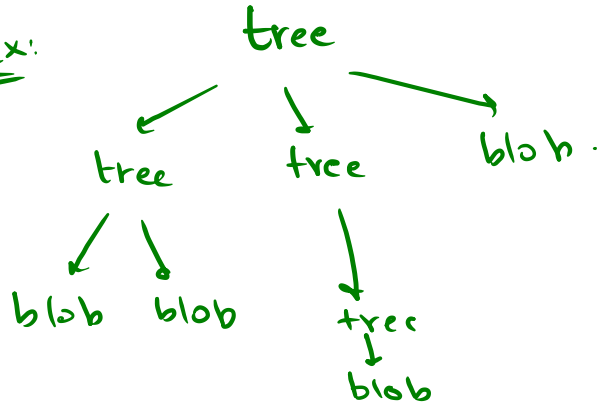
src
└ index.js

index.js

NOTE:

* To manage directory git. manages tree

ex:



$ git cat-file -p <hash> → spits content **

-t → type flag **

tree - used to manage direction

TYPES:-

* blob

-or-

* tree (once committed) → management of directories & sub directories

↓

stores refrences to root & file blobs.

-or-

* (commit) → "object"

COMMIT : Snapshot of code

- author
- msg (etc..)
- timestamp → due to timestamp makes unique commit id / hash !

after initial commit, next commits have link to parent commit obj.

o—o—o .... (commit & hash management)

* git optimizes by creating a pack file.

✓ does optimization → uses delta compression. (criteria) → pack files.
(during remote push) uploaded. **

* for code changes → hashes are created. ✓

optimizn → Same hash type stores @ same folder... ✓

$ git diff ??

What happens when we do $ git diff. → data is calculated OTG.

for large files → pack file optimizn.

To check contents of packfile. # $ git gc → garbage collection → pack file gen

$ git verify-pack -v .git/pack/pa.....

(compression here)

Stores all obj blobs ... etc...