Git

What is Git?

Git is a content-addressable system. At the core, it is a simple key-value data store. So you can insert any kind of file into a git repository, and it would give you a key to retrieve it later.

git hash-object

git hash-object is a plumbing command (used internally). It takes some data in, stores it in your .git/objects directory, and gives you back a unique key that now refers to that object.

Initially, the objects directory is empty. It has two sub-directories, pack and info inside it.

Working: git hash-object filename

- Returns a hash value used to call the object.
- Use -w flag to write the object as well to the database.
- The output is a 40-character checksum hash. SHA-1 hash.
- If -w was used, you'd now find an object with given hash value in the objects directory.

git cat-file

reverse of hash object. Creates a file from a hash value. Return the file.

Working: git cat-file hashvalue

Use -p to instruct git to first figure out type of object, then display it appropriately.

Use -t with cat-file to find the type of the object.

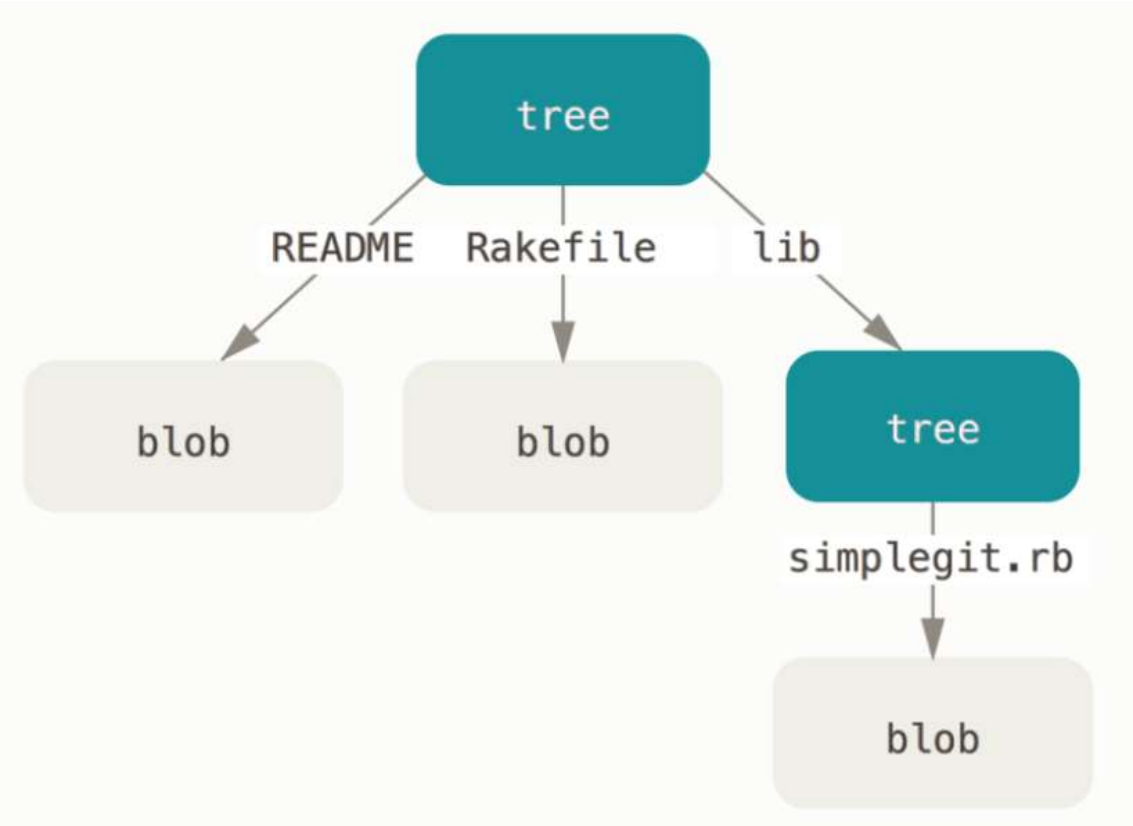
Using both in tandem

You can implement a basic version control using both these commands.

- Create a test file: echo 'test1' > test.txt
- Create hash value for the file: git hash-object -w test.txt
- It'd return a H1 hash for the file.
- Now change the value of file: echo 'test2' > test.txt.
- Create a hash value for the new file: git hash-object -w test.txt
- It'd return H2.
- Now you can delete the local copy of the test.txt
- Use git cat-file -p H1 > test.txt to retrieve a file with 'test1'.
- Use git cat-file -p H2 > test.txt to retrieve a file with 'test2'.

Tree Objects

All content is stored in git similar to UNIX filesystem, but simplified. All content is stored as tree and blob object types. Tree allows you to store multiple files.



Every file is stored as a blob, and multiple blobs make up a tree.

Creating your own tree

You can create your own tree. Git creates a tree by taking objects from the staging area and writing a series of tree objects from there. To do this, first set up an index. This can be done by plumbing command git update-index.

How to:

- Create an index with a single file: git update-index --add --cacheinfo filemode hashvalue filename.
- There are 3 filemodes: 100644 for normal file, 100755 for executable file, and 120000 for symbolic link.
- Use --cachefile to add an object which no longer exists and has to be pulled from the database.
- Create the tree with git write-tree. This would automatically write the file and return a hashvalue for the tree.
- You can read the current tree by using git read-tree.

Commit Objects

Different trees represent different snapshots of your project. Commit objects remembers the SHA-1 value of the trees. They also store that who made the commit, why the commit was made, and when they were saved.

To commit a tree, use commit-tree.

git commit-tree hashvalue

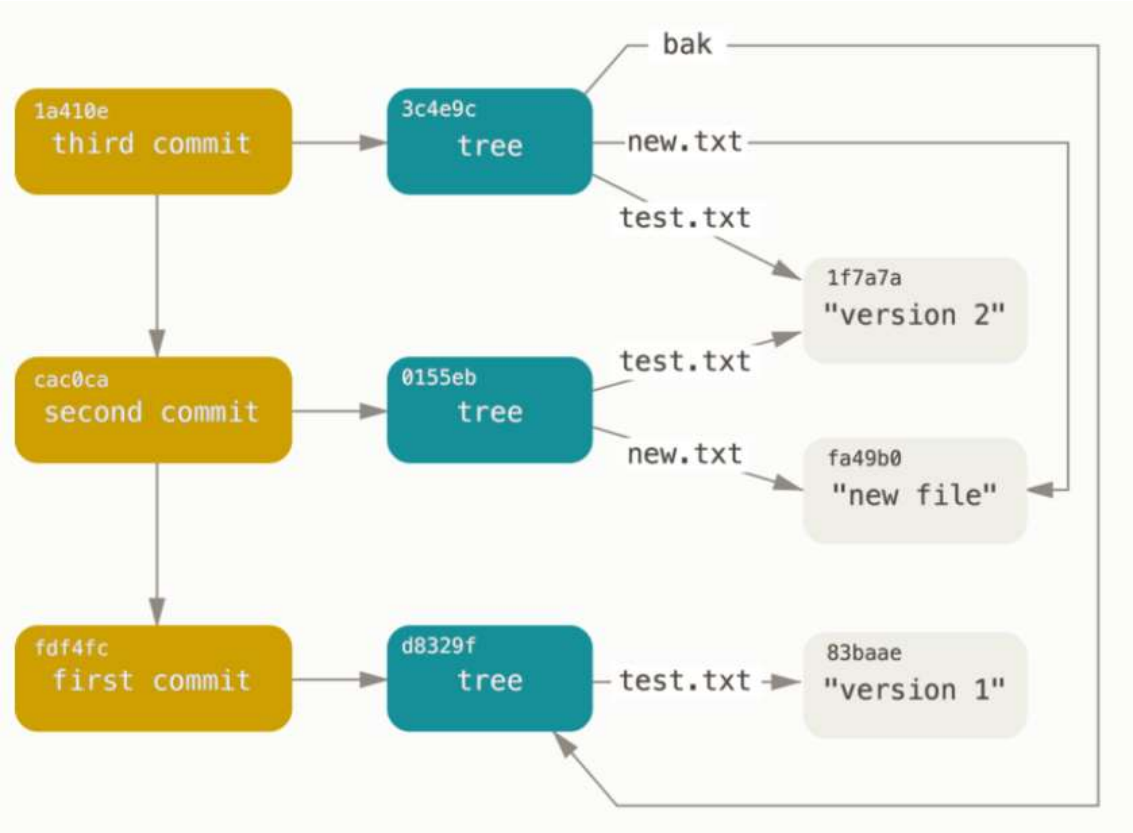
This would return a hashvalue for the commit object. You can commit multiple trees to have control versions of your project. Each commit object points to one of the snapshot tree you created.

You can view the commit history using git log

Git add and Git commit

They do exactly what is described earlier. It stores the blobs for the files that have changed, updates the index, writes out trees, and write commit objects that reference top-level trees and commit that came before them.

This looks like this:



Object storage

How git stores is best explained using an interactive example done on a string.

- Let's take the example for a string S.
- Git first constructs a header by identifying which type of object (in this case, a blob). Then after a blank space, the size of object is stored, and then a final null byte. For eg. Let S = "What is up, doc?", then it's header would be "blob 16\u0000".

- Then, the header and the content is appended to form a new string.
- The SHA-1 value of the new string is calculated.
- The new content is compressed with zlib library.
- The compressed content is written to the disk. The first two characters of the SHA-1 value determines the subdirectory name of the object, and last 38 give it the filename.