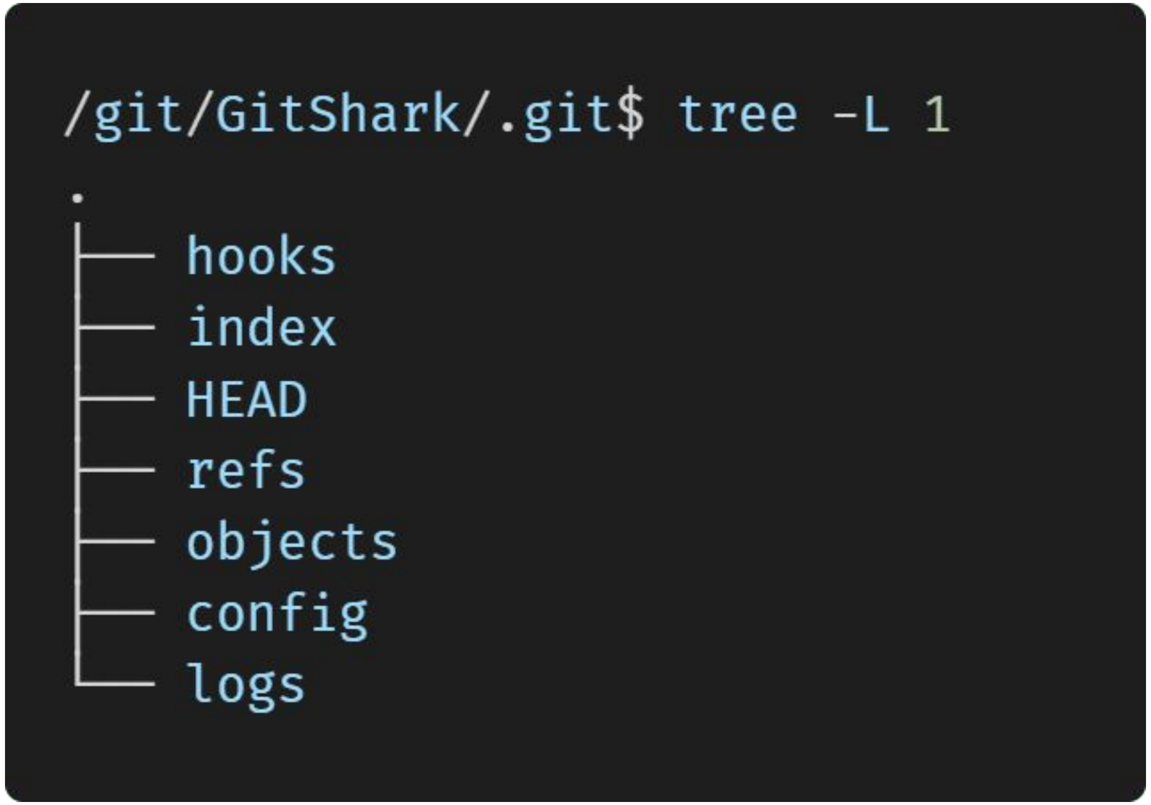
Git Internals

Divyansh Rajesh Jain

Anatomy of Git Repository



```
/git/GitShark/.git$ tree -L 1
.
├── hooks
├── index
├── HEAD
├── refs
├── objects
├── config
└── logs
```

Git Hooks

- Executable that runs before/after certain git actions (i.e. commit)
- Used for automation (i.e. formatting commit messages)
- Can be written in anything → has to be executable

Index

- Stores information about the “staging area”
- Stores the files that are staged (git add)
- It is in a binary format
- It is used to prepare a commit after the files have been staged

Config

- Stores global information about the repository
    - For example, the name of the author of the repository

Git Objects

- These are the “database” objects of a git repository
- It stores the “actual” information about the repository
- There are three basic git objects:
    - blob
    - tree
    - commit

Git Objects

- Indexed by the SHA-1 checksum
    - First two characters of checksum is the directory in objects
    - Remaining characters is the file name

Blob Object

- Contains the actual text data associated with a file

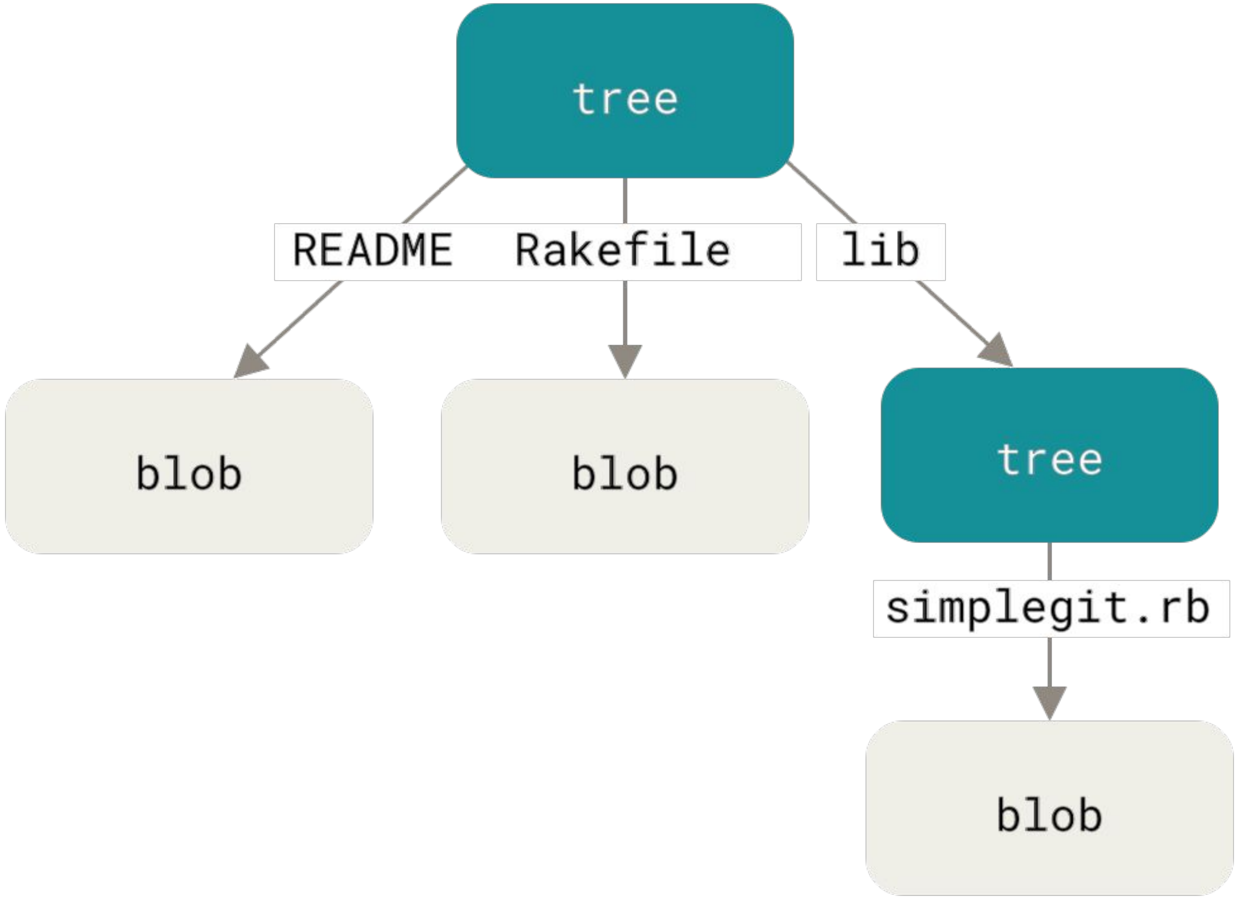
Tree Object

- Has one of more entries pointing to either other tree objects, or blob objects
- This is similar to UNIX filesystem but simpler

Tree Object Entries

- Every entry in a tree object stores the following:
  - Mode - 100644 (Normal File), 100755 (Executable File), 120000 (Symbolic Link)
  - Type - Either blob or tree
  - SHA-1 checksum
  - actual name of folder or file

Tree Object (Conceptual Repr)



Commit Object

- Stores following information for the commit
  - Author
  - Commit Message
  - Date and Time
  - SHA-1 checksum of a Tree Object that corresponds to that commit (full directory structure)
  - Parent Commit

Exercise

- What is the actual information that is stored in the .git directory?
- What is the difference between the three git object types?

But?

What is the problem with this scheme?

Motivation of Packfiles

Suppose there is only a few lines that have changed between two versions of a file?

Packfiles

- Optimization of git to merge git objects together
- Merges and stores diff of objects that are “similar” to each other?
- What makes two git objects “similar” to each other?

Packfiles

- Two files that have the same (similar) name which are of similar size are considered “similar”
- The base is stored, and all versions above (or potentially below) are stored as diffs.

Packfiles Storage

- There are two files in a packfile
    - Packfile: Contains the compressed version of the git objects that were compressed
    - Index: Index to the Packfile which gives the offset of each original git object (specificed by SHA-1 checksum of object)

Git Refs

- It is a way to store a symbolic link to a certain commit

- Special Type of Ref → Branches

Git Branches

- Each Branch Ref is stored in the ref/heads directory
- Each file in that directory has the name of the branch
- Each of these refs points to a specific commit object
  - That is how when you run: git checkout <branch> it knows the state of repository at that branch
- HEAD file in .git → Points to checked out branch ref

Git Logs Folder

- Stores all the changes in a repository in the log file
- It is useful for data recovery
  - In case of accidental reset → Object files not deleted!
  - Recover object SHA-1 from the logs folder and then reattach the ref of the branch to that object

Thank You!

Next Time: More on Remote Git Repositories