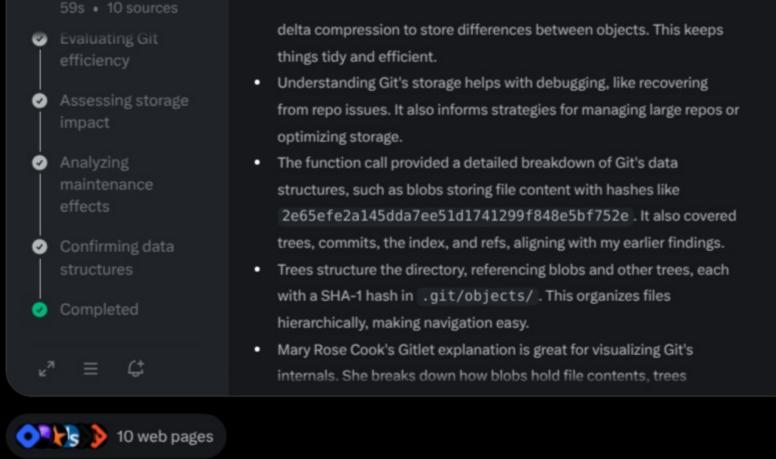
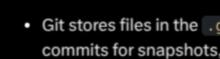
go in depth (search like a hell) to explain me how git exactly stores file in .git directory? how it uses data structure (include all technical jargons) I'm a Software Enginner, pretend yourself as sotware engineer as

use Elon Musk's first principals (Semantic Tree approach) to explain stuff!.



**Exploring Git storage** 



S DeepSearch

## Research suggests Git uses packfiles for further compression, which might be an unexpected detail

- for many users.
- **How Git Stores Files** Git, a distributed version control system, keeps the project history in the git directory. When you add

a file, Git creates a **blob** to store its content, identified by a unique SHA-1 hash. For example, a file with

.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4. When you commit, Git builds trees to represent the directory structure, referencing blobs and other trees, and creates **commits** that point to

"hello" might be stored as a blob with hash d670460b4b4aece5915caf5c68d12f560a9fe3e4 in

these trees, capturing the repository's state with metadata like author and timestamp.

## This approach is efficient because Git reuses blobs for identical content, saving space. An unexpected

research conducted.

Introduction

Trees:

blobs.

Storage Process and Workflow

.git/objects/.

2. Committing Changes (git commit):

reference existing trees.

References and Navigation

Branches:

Tags:

stored as a single blob, reducing redundancy.

Commits:

detail is that Git can also use packfiles for compression, storing differences between objects to optimize storage over time. For more details, check the official Git documentation on Git Objects.

A Comprehensive Survey Note on Git's File Storage in the .git Directory This note provides an in-depth exploration of how Git stores files in the .git directory, focusing on the

data structures and technical details. It aims to mimic the style of a professional article, offering a strict

superset of the information in the direct answer section, and includes all relevant details from the

## collaboration, version reversion, and branch management. The git directory, hidden within the project

Git, a distributed version control system, is designed to track changes to files over time, enabling

Git's storage model is built on three primary data structures, each serving a distinct purpose: Blobs (Binary Large Objects): Definition: Blobs store the raw content of files, without any metadata like filenames or paths. Storage: Each blob is identified by a SHA-1 hash of its content, ensuring content-addressable

#### Efficiency: Since blobs are based on content, identical files (even with different names or locations) share the same blob, enabling deduplication and space efficiency.

in the index (staging area).

other trees (subdirectories).

Structure: Each tree entry includes:

organizing the hierarchical structure.

leveraging specific data structures like blobs, trees, and commits.

**Core Data Structures and Storage Mechanisms** 

 Mode (permissions, e.g., 100644 for a regular file, 040000 for a directory). Type (blob or tree). SHA-1 hash of the referenced blob or subtree. Name of the file or directory.

Storage: Trees are also stored in .git/objects/, identified by their SHA-1 hashes, similar to

Example: A root tree might reference a blob for README.md and a subtree for a src directory,

Definition: Commits capture snapshots of the entire repository at specific points in time,

SHA-1 hash(es) of parent commit(s), enabling history tracking (multiple parents for merge

Creation: Blobs are created when files are added using git add, and their hashes are recorded

Definition: Trees represent the directory structure, acting as containers for blobs (files) and

- Structure: A commit object includes: The SHA-1 hash of the root tree it points to.
- location. Graph Property: Commits form a directed acyclic graph (DAG), where each commit points to its parent(s), supporting branching and merging.

Storage: Commits are stored in \_\_git/objects/ , with their SHA-1 hashes determining their

timestamp, message), and referencing the previous commit (parent). Both the tree and commit are stored in .git/objects/, identified by their SHA-1 hashes.

Git creates a tree object representing the current directory structure, using the blobs from the

 A special reference pointing to the current commit, usually through a branch. Stored in .git/HEAD, it might contain ref: refs/heads/main.

Stored in .git/refs/heads/, each branch is a file containing the SHA-1 hash of the commit it

 Similar to branches, stored in .git/refs/tags/, used to mark specific points in history, such as releases. These references allow developers to switch between different states (e.g., git checkout main)

can help recover lost commits or identify issues.

Run git commit -m "Initial commit":

Modify hello.txt to "hello again" and commit:

Description

filename, used

Stores file content, no

Stores

directory

structure,

entries with

or subtrees.

SHA-1 of blobs

contains

To illustrate, consider a simple repository with one file, hello.txt, containing "hello": Run git add hello.txt:

Creates a blob with hash d670460b4b4aece5915caf5c68d12f560a9fe3e4, stored in

Creates a tree referencing the blob, with hash, say, f2a3b4c5d6e7f8g9h0i1j2k3l4m5n6o7p.

Creates a commit pointing to this tree, with hash, say, a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6.

understand how commits, trees, and blobs interact, forming the DAG.

.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4.

Branch main (.git/refs/heads/main) points to this commit.

Index records: hello.txt d670460b4b4aece5915caf5c68d12f560a9fe3e4.

Creates a new blob with hash e69de29bb2d1d6434b8b29ae775ad8c2e48c5391.

Creates a new tree and commit, referencing the previous commit as parent.

The old blob remains in .git/objects/, demonstrating Git's reuse of unchanged content.

Storage

Location

.git/objects

.git/objects

fd/4f9947...

Understanding Git's internal storage model is crucial for advanced usage and troubleshooting:

Debugging: If a repository becomes corrupted, knowing where data is stored (e.g., .git/objects/)

 Performance Optimization: Large repositories benefit from Git's deduplication and packfiles, but engineers can optimize workflows by understanding these mechanisms, such as using git gc to

Advanced Operations: Concepts like rebasing, merging, and resetting are easier to grasp when you

- Blob d670460b4b4aece5915caf5c68d12f560a9fe3e4 (e.g., for version d6/70460b... control of file contents.
- metadata, .git/objects points to top-Commit (e.g., a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6 level tree, a1/b2c3d4... parent commits, author info. Reference Description Storage Location Example Type Points to current HEAD .git/HEAD ref: refs/heads/main commit, usually via branch. Points to a .git/refs/heads/ Branch commit, 7f54a437d87cd1f241cfb893c4823bc7e60c19ec (e.g., main) mutable. Marks specific commit, N/A (varies by tag name) Tag .git/refs/tags/ often for releases.
- model. Conclusion

Git's storage in the .git directory is a sophisticated system based on immutable objects (blobs, trees, commits) identified by SHA-1 hashes, organized to allow efficient storage, retrieval, and manipulation of

project history. This design supports collaboration, version control, and performance optimization,

These tables summarize the key components and their roles, enhancing understanding of Git's storage

## **Key Points** Git stores files in the .git directory using blobs for file contents, trees for directory structures, and commits for snapshots, all identified by SHA-1 hashes. It seems likely that this system allows efficient storage by reusing identical content across commits.

### folder, serves as the repository's backbone, storing the entire history and state of the project. This survey note delves into the mechanics of how Git stores files, using a content-addressable filesystem and

storage. The blob is stored in .git/objects/, with the first two characters of the hash forming a subdirectory and the remaining 38 characters forming the filename. For instance, a file with content "hello" might have a hash d670460b4b4aece5915caf5c68d12f560a9fe3e4, stored in .git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4.

- - forming the history graph.
- commits). Author and committer information (name, email, timestamp). Commit message for documentation.
- The process of storing files in Git involves several steps, reflecting its operational workflow: Adding Files (git add):

When a file is added, Git reads its content, computes a SHA-1 hash, and stores it as a blob in

The blob's hash is recorded in the index ( .git/index ), which maps file paths to their

A commit object is then created, pointing to the root tree, including metadata (author,

Tree reuse occurs when directories remain unchanged between commits, allowing Git to

corresponding blob hashes. This staging area prepares files for commit.

index. This tree recursively references other trees for subdirectories.

3. Efficiency Through Reuse: Git's content-addressable design ensures content deduplication: identical file contents are

Over time, Git can optimize storage by creating packfiles, which compress multiple objects into

a single file using delta compression. This stores differences between objects (e.g., diffs between file versions) rather than full copies, enhancing efficiency for large repositories.

HEAD:

Git uses references to navigate and manage the repository, complementing the object storage:

points to. For example, .git/refs/heads/main might contain

7f54a437d87cd1f241cfb893c4823bc7e60c19ec.

without directly manipulating SHA-1 hashes, enhancing usability. **Practical Implications for Software Engineers** 

pack objects.

Example Workflow

Object

Type

Tree

Tables for Clarity To organize the data structures and their properties, consider the following tables:

Example Hash

f2a3b4c5d6e7f8g9h0i1j2k3l4m5n6o7p

Stores snapshot

(e.g.,

# making Git a powerful tool for software development.

- **Key Citations** Git Internals Git Objects detailed explanation
- How Git stores data comprehensive guide Git from the inside out detailed walkthrough