A Deep Dive into Git: Understanding What's Under the Hood

Objects, Branches, and How to Create a Repo From Scratch

Introduction

- Git is an essential tool for version control.
- We often use Git commands without understanding the underlying mechanisms.
- This presentation aims to demystify Git by exploring its core concepts.
- By the end, you'll have a solid grasp of how Git works.

Objectives

- Understand Git objects: blobs, trees, and commits.
- Explore branching and its implementation.
- Dive into the working directory, staging area, and repository.
- Create a repository from scratch, deepening our Git knowledge.

Git Objects - Blob, Tree, and Commit

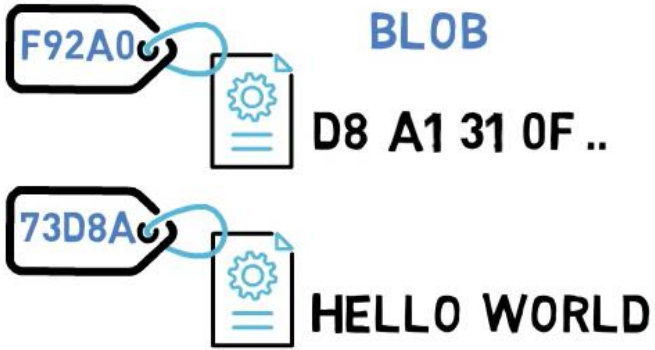
- Git objects are the building blocks of version control.
- Blobs store file contents as binary data.
- Trees represent directories and their contents.
- Commits are snapshots of the entire project at a point in time.

Git as a Snapshot System

- Git maintains a snapshot of your project at each commit.
- Imagine it as a timeline of your project's state.
- Each commit points to a specific snapshot in time.
- This allows you to revisit any point in your project's history.

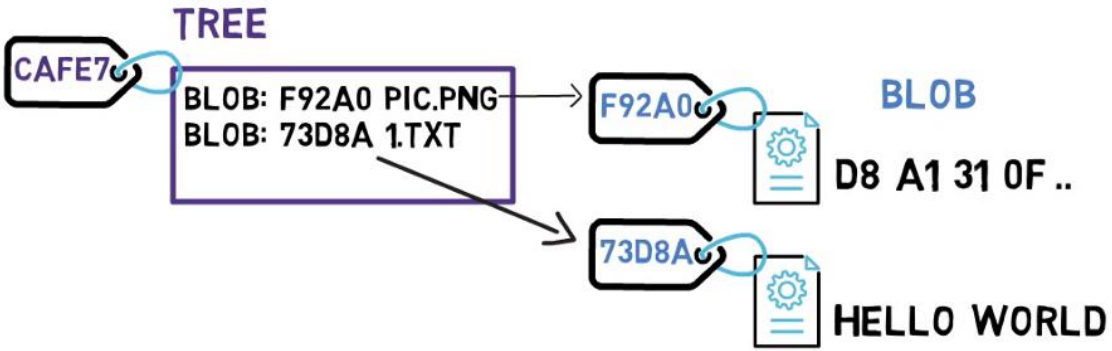
Blobs in Git

- Blobs store the content of files.
- They are identified by SHA-1 hashes.
- Unlike files, blobs lack metadata like creation dates.
- Blobs are fundamental for tracking file changes.

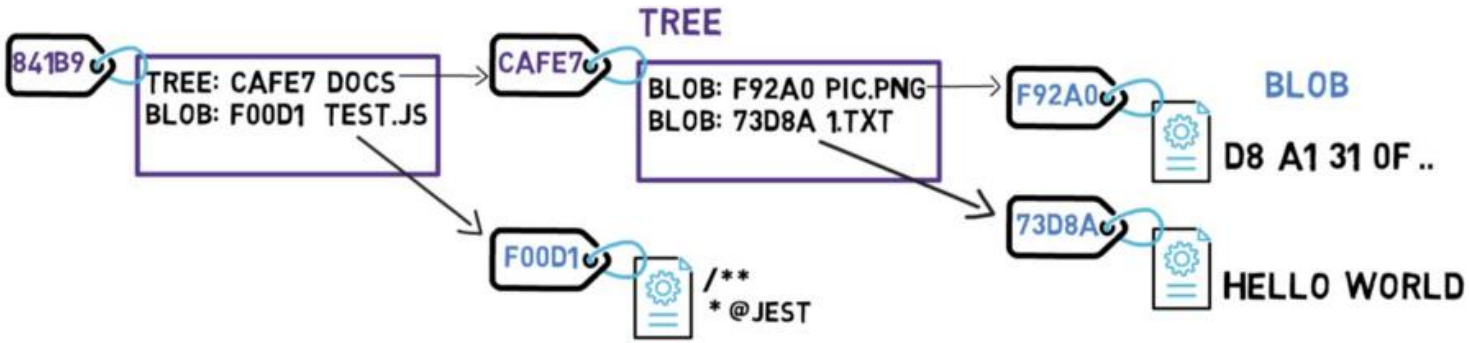


Trees in Git

- Trees act like directories in a file system.
- They point to blobs and other trees.
- Each tree is identified by a SHA-1 hash.
- Trees organize the structure of your project.



Trees in Git

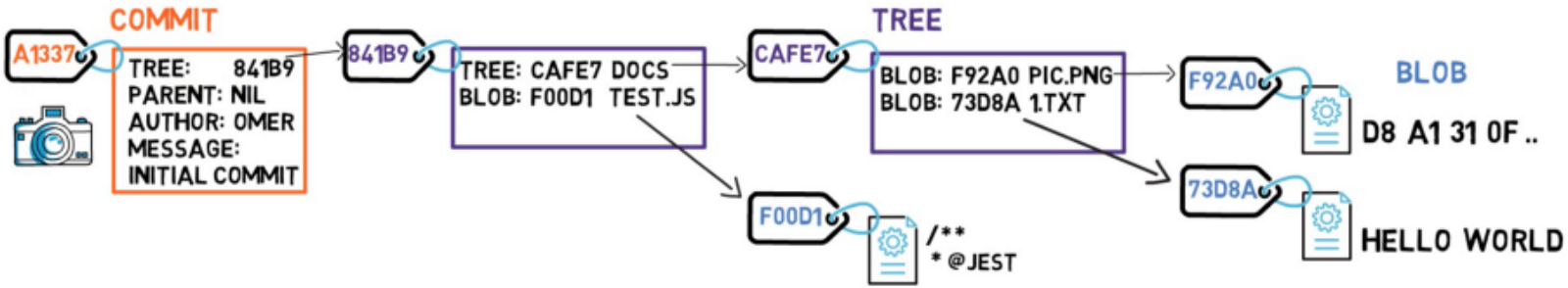

841B9
TREE: CAFE7 DOCS
BLOB: F00D1 TEST.JS
CAFE7
TREE
BLOB: F92A0 PIC.PNG
BLOB: 73D8A 1.TXT
F92A0
BLOB
D8 A1 31 0F ..
73D8A
HELLO WORLD
F00D1
/**
* @JEST

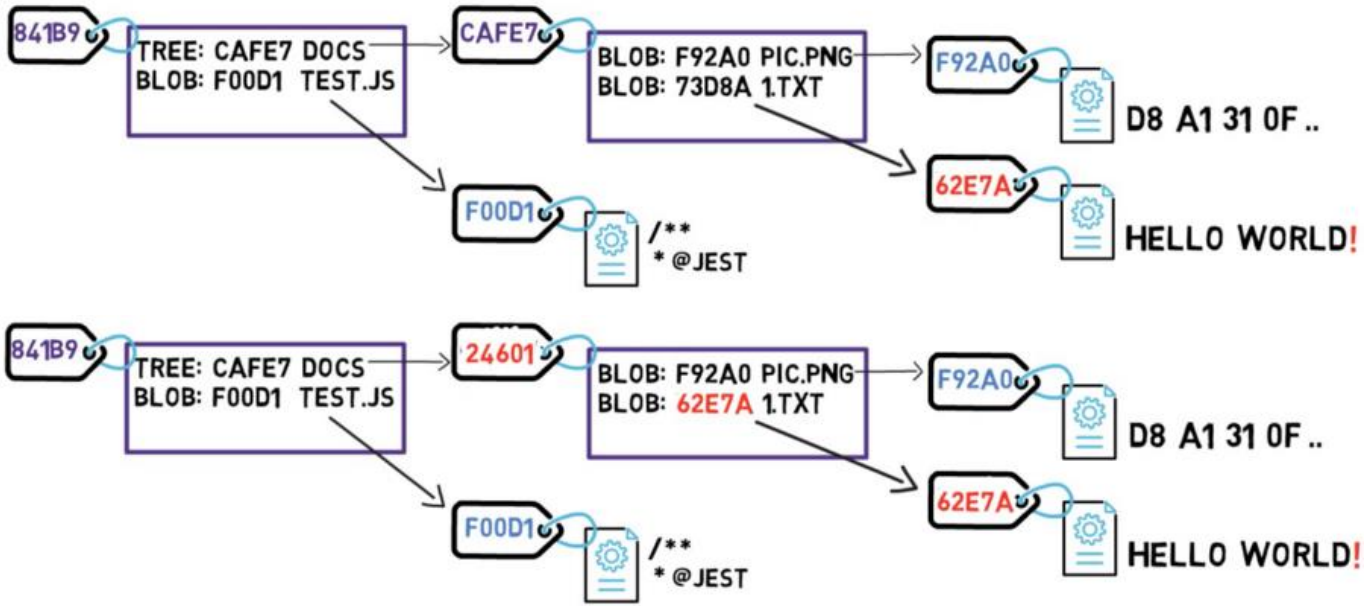
Creating a Snapshot in Git (Commit)

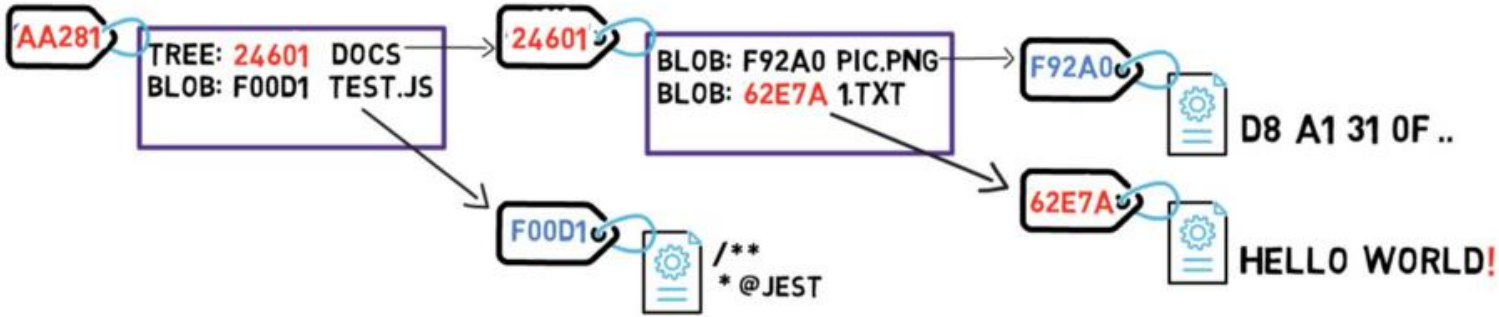
- Commits capture the entire project state.
- They include a pointer to the root tree.
- Commit metadata contains the committer's details and a message.
- Most commits have one or more parent commits.

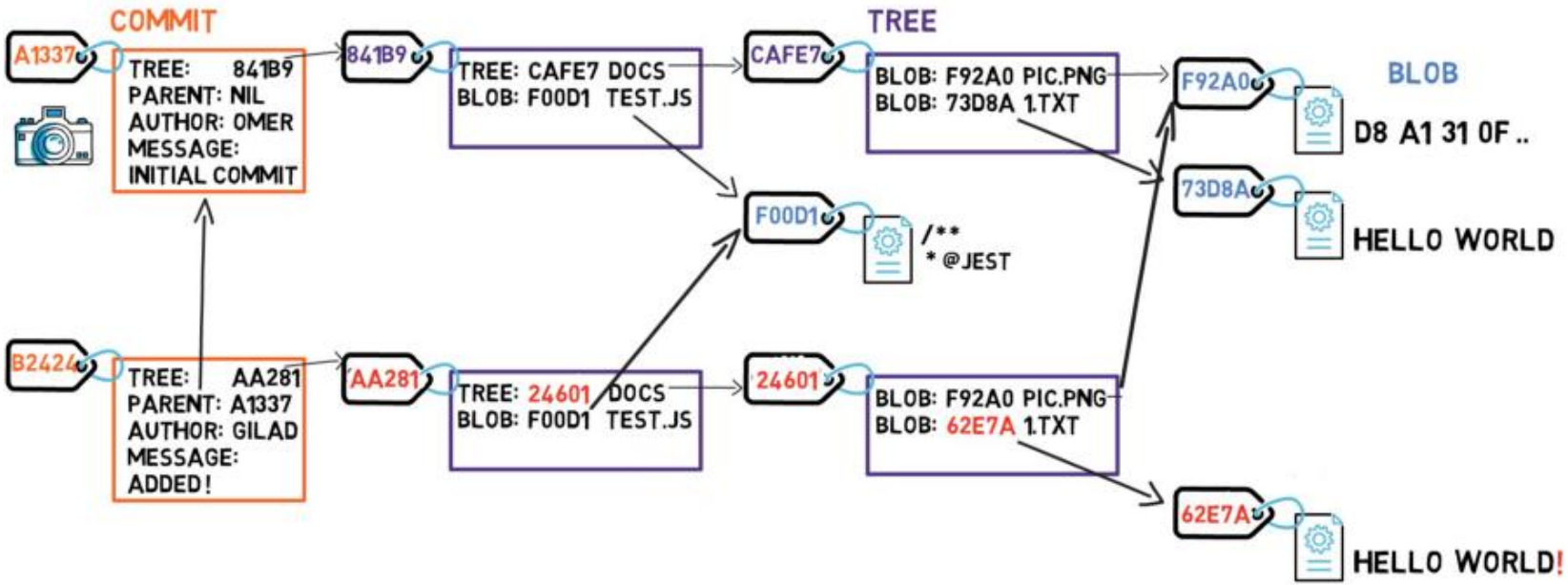
Commit Objects in Git

- Commits are uniquely identified by SHA-1 hashes.
- Each commit contains a full snapshot of your project.
- Commit objects include details like timestamps.
- Parent commits create a history of changes.




841B9
TREE: CAFE7 DOCS
BLOB: F00D1 TEST.JS
CAFE7
BLOB: F92A0 PIC.PNG
BLOB: 73D8A 1.TXT
F92A0
D8 A1 31 0F ..
62E7A
HELLO WORLD!
F00D1
/**
* @JEST
841B9
TREE: CAFE7 DOCS
BLOB: F00D1 TEST.JS
24601
BLOB: F92A0 PIC.PNG
BLOB: 62E7A 1.TXT
F92A0
D8 A1 31 0F ..
62E7A
HELLO WORLD!
F00D1
/**
* @JEST


AA281
TREE: 24601 DOCS
BLOB: F00D1 TEST.JS
24601
BLOB: F92A0 PIC.PNG
BLOB: 62E7A 1.TXT
F92A0
D8 A1 31 0F ..
62E7A
HELLO WORLD!
F00D1
/**
* @JEST


COMMIT
A1337
TREE: 841B9
PARENT: NIL
AUTHOR: OMER
MESSAGE:
INITIAL COMMIT
841B9
TREE: CAFE7 DOCS
BLOB: F00D1 TEST.JS
CAFE7
TREE
BLOB: F92A0 PIC.PNG
BLOB: 73D8A 1.TXT
F92A0
BLOB
D8 A1 31 0F ..
73D8A
HELLO WORLD
F00D1
/**
* @JEST
B2424
TREE: AA281
PARENT: A1337
AUTHOR: GILAD
MESSAGE:
ADDED!
AA281
TREE: 24601 DOCS
BLOB: F00D1 TEST.JS
24601
BLOB: F92A0 PIC.PNG
BLOB: 62E7A 1.TXT
62E7A
HELLO WORLD!

Efficient Data Storage in Git

- Git optimizes data storage by avoiding redundancy.
- Unchanged objects aren't duplicated; they're referenced.
- This efficient storage keeps repository sizes manageable.
- Commits only store changes, not the entire project.

Git Object Recap

- Git has three primary objects: blobs, trees, and commits.
- Blobs store file content, trees represent directories, and commits capture snapshots.
- Each object is identified by a unique SHA-1 hash.
- Git's efficiency comes from referencing unchanged objects.

Object Hash Consistency

- Objects with identical content generate the same SHA-1 hash.
- This hash consistency ensures data integrity.
- It's why two identical blobs or trees have the same hash.
- Commits may have different hashes due to metadata.

Branches in Git

- Branches are named references to specific commits.
- 'Master' is a common default branch name.
- Git uses branch names to track lines of development.
- Creating a branch is like adding a label to a commit.

Main Development Line - 'Master' Branch

- 'Master' is often the primary development branch.
- It's just a convention; you can name it differently.
- The 'master' branch typically tracks the latest progress.
- Think of it as the default starting point for your project.

Creating a New Branch

- To create a new branch, you're essentially adding a label.
- The new branch points to the same commit as the current one.
- You can name branches according to your workflow.
- Branches allow for parallel development.

HEAD Pointer in Git

- HEAD is a crucial pointer in Git.
- It indicates the currently active branch or commit.
- HEAD usually points to a branch, which points to a commit.
- It's essential for understanding your working context.

Switching Between Branches

- Switching branches changes where HEAD points.
- Use 'git checkout' to change the active branch.
- This allows you to work on different branches.
- Each branch represents a unique line of development.

Recording Changes in Git

- Git tracks changes in your project.
- The working directory is where you make edits.
- Changes are first registered in the index (staging area).
- Commits are created based on the state of the index.

The Role of the Index

- The index allows precise control over commits.
- It's also called the staging area.
- You can selectively stage changes using 'git add.'
- Committing only staged changes helps maintain a clean history.

Tracked vs. Untracked Files

- Git categorizes files as tracked or untracked.
- Tracked files are either in the last snapshot or staged.
- Untracked files are not part of the repository's history.
- Understanding this distinction is crucial for managing changes.

Thanks You!