AAiT
ADDIS ABABA INSTITUTE OF TECHNOLOGY
አዲስ አበባ ቴክኖሎጂ ኢንስቲትዩት
ADDIS ABABA UNIVERSITY
አዲስ አበባ ዩኒቨርሲቲ

Department of Software Engineering

Data Structure and Algorithm Group Project

Title: MiniGit, A Custom Version Control System

Group Members	ID	User Name
1. Fitsum Nigusse	ATE/4123/15	fitse1011
2. Mussie workneh.	ATE/9096/15	Mo-me-0
3. Mihretu Tesema	ATE/1022/15	nebiyutesema
4. Peter Koru	ATE/1191/15	p1111kr
5. Yonathan Tatek	ATE/6955/15	joniman478

Submitted to: Dr. Beakal Gizachew

Submission Date: 22/06/2025

MiniGit: A Custom Version Control System

MiniGit is a simplified, local-only version control system developed in C++. It is designed to replicate the core functionalities of Git, including version tracking, branching, committing, merging, and viewing logs. The objective of this project is to deepen our understanding of how modern version control systems work under the hood using fundamental data structures and file operations.

Core Concepts

➢ Repository (.minigit/): The main directory where MiniGit stores all version control information, including commit objects, blob objects, the staging area, and references (branches).

➢ Objects (.minigit/objects/):

Blobs: Represent the content of files at a specific point in time. Each unique file content is stored as a blob, identified by its hash.

Commits: Snapshots of the entire project at a given time. A commit object stores metadata (message, timestamp, and commit hash), a pointer to its parent commit (for history), and a list of file paths mapped to their corresponding blob hashes. Commit objects are also identified by their commit hash.

➢ Staging Area (.minigit/index): A temporary file that records the files and their versions that are prepared to be included in the next commit. Files are "added" to this area.

➢ HEAD (.minigit/HEAD): A special reference that points to the current commit. It can point directly to a commit hash by default it contain ref: refs/heads/main.

➢ Branches (.minigit/refs/heads/): Named pointers to specific commit. They allow developers to work on different lines of development concurrently. Each branch is a file whose content is the hash of the commit it points to.

Features

1. Initialization (init)

   - √ Creates a hidden `.minigit/` directory.
   - √ Initializes internal data structures for commits, branches, and object storage.
   - √ Sets the default branch to `master`.

2. Adding Files (add <filename>)

   - √ Checks if a file exists and stages it for the next commit.
   - √ Reads the file content and hashes it using a custom hash function based on it's content.
   - √ Stores the content in `.minigit/objects/<hash>` directory.

3. Committing (commit -m <message>)

   - √ Captures a snapshot of all staged files.
   - √ Records metadata such as timestamp, commit message, and parent commit.
   - √ Stores commits in objects and updates the current HEAD accordingly.
   - √ Adds each commit to a linear commit list for traversal.

4. Viewing Log (log)

   - √ Displays the commit history in reverse chronological order(from recent to older commits).
   - √ Lists commit hash, message, and timestamp for each commit.

5. Branching (branch <branch-name>)

   - √ the branch command has two functionalities: With no argument taken, it displays the branchs highlighting the current branch, and with argument, it creates a new branch pointing to the current commit.

6. Checkout (checkout <branch-name> or checkout <commit-hash>)

√ Switches the working directory to the specified branch or commit.

√ Updates the HEAD reference accordingly.

7. Merging (merge <branch-name>)

√ Finds the lowest common ancestor between the current and target branches.

√ Performs a three-way merge based on file versions.

√ Identifies and reports conflicts, if thete are any and display's it.

8. Diff Viewer (diff <file1> <file2>)

√ takes two files and compare them lene by line to identify differences

√ displays the different parts indicated with there line number

Project Structure

The MiniGit project consists of the following C++ files:

➢ main.cpp: The entry point of the application. It parses command-line arguments and dispatches commands to the MiniGit class.

➢ MiniGit.cpp: Contains the MiniGit class, which encapsulates the core logic of the version control system. This includes methods for initialization, adding files, committing, viewing logs, branching, and checking out.

➢ fileUtils.cpp: Provides utility functions for file system operations, such as checking file existence, creating directories, reading/writing file content, and removing files.

➢ commitList.cpp: Defines the CommitNode structure and the CommitList class.

CommitNode: Represents a single commit in the version history, storing its hash, timestamp, message, a pointer to its parent commit, and a map of file paths to blob hashes.

CommitList: Manages a linked list of CommitNode objects (in-memory representation of commit history) and provides methods for adding commits and serializing/deserializing CommitNode objects to/from disk format.How to Use

Setup and Compilation

➢ Compile: g++ minigit_fileutils.cpp fileutils.cpp -o minigit

➢ Run: ./minigit <command> <argument/s>

➢ Commands:

√ `init` - Initialize the repository.

√ `add <filename>` - Stage a file.

√ `commit -m "message"` - Create a new commit.

√ `log` - Show commit history.

√ `branch <name>` - Create a branch.

√ `checkout <name/hash>` - Move to a branch or commit.

√ `merge <branch>` - Merge a branch.

√ 'diff <file1> <file2>' - Identify the differences between two files

Limitations

➢ No remote repository support.

➢ Basic conflict handling in merge (only message output).

➢ Limited error handling

➢ Linear commit history maintained; no visualization.

Future Improvements

➢ Implement more file diffing tool for `diff` command

- Add a GUI wrapper for visualization
- Online repository option
- Author and collaboration
- Improve merge conflict resolution with markers