ADDIS ABABA INSTITUTE OF TECHNOLOGY

Software engineering department

Fundamental of Data Structure and Algorithm Analaysis

(SECT-3091)

Project Title: MiniGit: A Custom Version Control System

Name	Id
1. Getamesay Hailemichael……......	ATE/5152/13
2. Sisay Leykun.....	ATE/0493/15
3. Tamrat Arage.....	ATE/8888/15

GitHub Repository: https://github.com/Tamrat16/DSA-Project-2-MiniGit-A-Custom-Version-Control-System

Submitted to**: Dr. Beakal Gizachew Assefa**

Submissin date: June 24, 2025

Table of Contents

1. Project Objective

MiniGit is a lightweight version control system that implements core Git functionality including file tracking, commits, branching, merging, and history management. The system uses fundamental data structures and algorithms to provide a complete version control experience.

2. Overview of MiniGit Features

- Repository Initialization
- Staging Area for File Tracking
- Commits with Messages and Timestamps
- Branch Creation and Checkout
- Commit History Logging
- Line-by-Line Difference Viewer (Diff Viewer)

3. System Design Overview

MiniGit is entirely file-based and organizes its data in a hidden directory called `.minigit` inside the working directory. The core components are:

3.1 Directory Structure

- `.minigit/`
  - `objects/` → stores file blobs (actual file contents)
  - `commits/` → stores commit metadata
  - `refs/` → stores branch references
  - `HEAD` → tracks the currently active branch

3.2 Key Functional Modules

- **StagingArea** → Manages files prepared for committing
- **Blob** → Handles file contents and hashing
- **Commit** → Manages commit creation and storage
- **Branch** → Manages branch creation and references
- **MiniGitSystem** → The main controller class handling all user commands

4. Data Structures Used

DSA Concepts	*Description*	*Feature*
File I/O, Directory structures	Repository initialization	init
Hash functions, Hash maps	File staging with hashing	add
DAGs, Linked lists	Snapshot creation	commit
Tree traversal, DFS	History traversal	log
Hash maps, References	Branch management	branch
Tree operations	State switching	checkout
BFS, Conflict resolution	Three-way merge	merge
Set operations	Repository state	status

5. Module Descriptions

5.1 StagingArea Module

- Tracks files added by the user for the next commit.
- Prevents duplicate entries using a `std::set`.
- Files remain staged until committed.

5.2 Blob Module

- Computes hash for file content using `std::hash`.
- Stores file contents in `.minigit/objects/` folder under their hash name.
- Retrieves file content based on hash during restoration or diff comparison.

5.3 Commit Module

- Stores each commit with parent reference, commit message, timestamp, and file mapping.
- Generates commit hash using the full commit data string.
- Saves commit files in `.minigit/commits/`.

5.4 Branch Module

- Creates branches as simple text files under `.minigit/refs/`.
- Each branch file contains the latest commit hash for that branch.

5.5 MiniGitSystem Module

- Main control center of the application.
- Interprets user commands and calls appropriate modules.

- Handles repository initialization, adding files, creating commits, logging, branching, checkout, and diffing.

6. Functional Workflow

6.1 Initialization (`init` command)

- Creates `.minigit` folder structure.
- Initializes `HEAD` to point to `master` branch.

6.2 Adding Files (`add <filename>` command)

- Stages a file for committing.
- Verifies file existence before staging.

6.3 Committing (`commit` command)

- Reads staged files.
- Computes hashes and stores them as blobs.
- Creates commit file containing parent commit, message, timestamp, and file hash map.
- Updates branch reference.

6.4 Viewing Logs (`log` command)

- Traverses commit history starting from current branch reference.
- Displays commit hash, timestamp, and message for each commit.

6.5 Branching (`branch <branchname>` command)

- Creates a new branch file pointing to current commit hash.

6.6 Checkout (`checkout <branch/commit>` command)

- Restores files to the state of specified commit or branch.
- Updates `HEAD` if switching branches.

6.7 Diffing (`diff <commit1> <commit2>` command)

- Compares file contents between two commits.
- Displays line-by-line differences.
- Handles added, removed, and modified lines.

7. Design Decisions

7.1 File-Based Storage

Using simple file storage (no database) allows students to easily inspect and understand internal repository structures.

7.2 Simplified Hashing

- `std::hash` used for quick hash generation.
- Simplifies implementation while introducing the concept of content-based addressing.

7.3 Plaintext Metadata

- Commits and branch references are stored as plain text files.
- Keeps system transparent and simple for educational inspection.

7.4 Minimal Error Handling

- Project focuses on core functionality.
- Robust error handling can be added as future improvement.

8. Limitations

Limitation	Description
No Merge Conflict Handling	Merge feature is not fully implemented.
No File Deletion Support	Cannot remove tracked files yet.
No Unstage Functionality	Cannot unstage files once added.
No Remote Repository Support	Works locally only.
No Real Hash Functions	Uses `std::hash` instead of secure hash functions like SHA-1.
Limited Performance Optimization	Reads entire commit history repeatedly; inefficient for large repositories.
No User Authentication	Commits have no user metadata (name, email, etc.).

9. Sample Use Case Demonstration

```
MiniGit > init
MiniGit > add test.txt
MiniGit > commit
Enter commit message: Initial commit
MiniGit > log
MiniGit > branch feature1
MiniGit > checkout feature1
```

```
MiniGit > diff <commit1> <commit2>
MiniGit > exit
```

10. Visual Representation of Repository Structure

```
.minigit/
  |
  |- HEAD
  |- refs/
  |    |- master
  |    |- feature1
  |
  |- commits/
  |    |- <commit-hash-1>
  |    |- <commit-hash-2>
  |
  |- objects/
       |- <blob-hash-1>
       |- <blob-hash-2>
```

11. Future Improvements

Feature	Description
Merge Support	Handle conflict resolution automatically.
File Deletion	Allow deleting tracked files safely.
Staging Control	Add `unstage` command to remove files from staging area.
Secure Hashes	Replace `std::hash` with SHA-1 or SHA-256.
Remote Support	Simulate push/pull to/from remote repository.
Status Command	Show file changes compared to last commit.
Visual Commit Graph	Display branch structure visually like real Git.
Compression	Compress blob contents to save disk space.
User Profiles	Track commits by user with name and email.
Improved Error Handling	Add meaningful error messages and recovery options.

12. Technologies Used

- Language: C++17
- Compiler: MSVC (Visual Studio)

- File System: `std::filesystem`
- Hashing: `std::hash`
- Input/Output: Standard I/O Streams

13. Conclusion

MiniGit successfully demonstrates the core concepts behind modern version control systems using simple C++ implementations. While significantly simplified compared to real-world Git, MiniGit serves as an effective learning tool to grasp the fundamental ideas of versioning, data integrity via hashing, and repository management.

With incremental improvements, MiniGit can evolve into a fully-featured Custom Version control System. The project lays the groundwork for future extensions such as merge conflict resolution, remote repository synchronization, secure hashing, file compression, and enhanced user interfaces.