College of Technology and Built Environment
School of Information and technology Engineering

Fundamentals of Data Structure and Algorithm Analysis

MiniGit Project Documentation

Group Members	ID
1. Edlawit Sewasew	UGR /5208/16
2. Habiba Ziyad	UGR/1844/16
3. Kaleb Nekatibeb	UGR/4300/16
4. Mikias Endalkachew	UGR/6414/16
5. Yonas G/Michael	UGR/8404/16

Submitted to: Mr. Beimnet

Submission Date: 20/06/2025

1. Data Structures Used

In designing MiniGit, various C++ Standard Template Library (STL) data structures were employed to represent key components of a version control system. The use of appropriate data structures was essential for simulating git-like behaviour effectively:

1. std::map<std::string, std::string> (File Map in Commits):

- Each commit stores a mapping of filenames to the hash of their blob content.

- This allows O(log n) lookups when comparing file states during operations like merge and diff.

2. std::vector<std::pair<std::string, std::string>> (Staging Area):

- Represents the temporary storage of files to be committed.

- Parsed from the "staging.txt" file, this structure efficiently supports iterative operations for committing staged files.

3. std::vector<Commit> and std::map<std::string, Commit> (Commit History):

- Commits are parsed from a text log into a vector for linear scanning (used in log) or a map for efficient ID-based access (used in merge/diff).

- The Commit struct includes:

  • id: Unique commit hash

  • time: Timestamp of the commit

  • message: Commit message

  • parent: ID of the parent commit

  • files: A map of filenames to their content hashes

4. std::set<std::string> (File Comparison in Diff):

- Used to compute the union of filenames across two commits for change detection.

5. std::filesystem:

- Handles directory and file operations (e.g., checking existence, reading blobs, writing content).

- Ensures that MiniGit works reliably across systems supporting C++17 and above.

2. Design Decisions

Several important design choices were made to keep MiniGit both understandable and structurally similar to actual Git, while maintaining educational clarity and simplicity:

1. Text-Based Repository Structure:

- All project metadata is stored in ".minigit/" as plain text files ("commits.txt", "branches.txt", "HEAD.txt", etc.).

- This allows transparency and easy debugging without requiring external tools.

2. Branching Design:

- "HEAD.txt" stores only the name of the current branch.

- "branches.txt" maps branch names to their latest commit ID.

- This follows Git's actual model of symbolic references.

3. Commit Format:

- Each commit is a serialized block in "commits.txt" including metadata and file snapshot mappings.

- Using append-only strategy avoids overwriting past data.

4. Blob Management:

- Each file version is hashed using a simple custom hash and stored in ".minigit/objects/".

- The filename is the hash itself, emulating Git's object model.

5. Simplified Merge and Conflict Strategy:

- Only the latest commits from both branches are compared.

- If the same file differs, it reports a conflict but keeps the current branch version.

6. Simplified Diff Implementation:

- Compares files line-by-line by loading blobs into vectors.

- Doesn't use complex diffing algorithms for simplicity.

3. Limitations and Future Improvements

While MiniGit provides a functional simulation of version control basics, it has limitations that suggest directions for improvement:

1. No Support for Binary Files:

- Currently optimized for text files; binary handling would require additional encoding or byte-safe comparison.

2. Simplistic Hashing Algorithm:

- Uses a basic custom function for content hashing.

- Should be replaced with a stronger hash like SHA-1 or SHA-256 to reduce collision risk.

3. Conflict Handling in Merge:

- Reports conflict but does not offer user interaction or automatic resolution.
- Future versions could allow user to choose between versions.

4. No Real History DAG:

- Merges only store one parent.
- Git-style DAG support with multiple parents for merge commits is a valuable extension.

5. No Undo, Reset, or Revert Support:

- These Git features are not present in MiniGit.
- Implementing reset/revert would enhance safety and usability.

6. No Push/Pull Functionality:

- This is a purely local system.
- Adding networking or shared folders would help simulate collaborative version control.