Contents

[MiniGit – Project Report 2](#_Toc201492321)

[Project Title: MiniGit – A Custom Version Control System 2](#_Toc201492322)

[1. Data Structures Used 2](#_Toc201492323)

[2. Real SHA-1 Integration 3](#_Toc201492324)

[3. Design Decisions 3](#_Toc201492325)

[4. Limitations 4](#_Toc201492326)

[5. Future Improvements 4](#_Toc201492327)

[6. Technologies Used 4](#_Toc201492328)

[7. File Structure 5](#_Toc201492329)

# **MiniGit – Project Report**

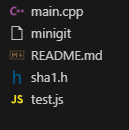
## Project Title: MiniGit – A Custom Version Control System

A simplified Git-like version control system implemented in C++, supporting file tracking, commits, branching, merging, and diffs, all from scratch.

### Data Structures Used

|  |  |  |
| --- | --- | --- |
| **Component** | **Data Structure** | **Purpose** |
| **Blob Store** | File-based (objects/<sha1>) | Stores snapshots of file content using SHA-1 hashes |
| **Staging Area** | index.txt (plain text) | Tracks files staged for the next commit |
| **Commits** | One file per commit in commits/ | Stores metadata: parent hash, timestamp, message, and file mappings |
| **Commit History** | Linked List (via parent hashes) | Used in log, merge, and diff to walk backwards through history |
| **Branches** | Map (branches/<name>.txt) | Maps branch names to their latest commit hash |
| **HEAD** | Pointer (HEAD.txt) | Points to current branch (ref: <name>) or detached commit hash |

### Real SHA-1 Integration



The system now uses a custom-built SHA-1 implementation (sha1.h) to:

* Hash file contents into blob identifiers (like Git)
* Generate commit hashes based on message + timestamp + file list
* Ensure unique and traceable commits and file snapshots

This ensures MiniGit mimics real Git’s hash-based content tracking, improves uniqueness and prevents collisions seen in simple hash functions.

### Design Decisions

* **Filesystem-Backed Storage**:  
  All commit metadata, file snapshots, and branch info are stored as plain text files inside .minigit/. This keeps it transparent, inspectable, and platform-independent.
* **Real SHA-1 Hashing**:  
  Provides collision-resistant content-addressing, improving integrity and traceability of blobs and commits.
* **CLI-Like UX**:  
  Commands closely follow Git's syntax (init, add, commit, log, branch, checkout, merge, diff), making it intuitive for developers.
* **Manual Conflict Handling**:  
  Merge conflicts are marked in files using standard Git-style conflict markers, allowing manual resolution before recommitting.

### Limitations

* No directory or subfolder support — only flat files are handled.
* No rename or file deletion tracking — files must exist in working directory to be tracked.
* No user identity in commits (e.g., author name/email).
* Conflicts are not automatically resolved — user must manually edit and commit.
* No remote repositories or network features.

### Future Improvements

* Track file renames and deletions
* Add support for directory trees and recursive tracking
* Include user metadata (e.g., author name/email) in commits
* Implement push/pull with local remotes
* Improve merge strategy with auto-resolution algorithms
* Visual commit graph with branch highlighting
* Add unit tests and edge case protection

### Technologies Used

|  |  |
| --- | --- |
| **Area** | **Tool / Language** |
| Programming | C++17 |
| CLI | Standard C++ I/O |
| Hashing | Custom SHA-1 (sha1.h) |
| Persistence | Filesystem API |
| Platform | Cross-platform (Linux/Windows) |

### File Structure

project/

├── main.cpp ← Main CLI logic

├── sha1.h ← Custom SHA-1 implementation

├── README.md ← Project documentation

├── .minigit/ ← Repository data folder (auto-created)

│ ├── HEAD.txt ← Current branch or commit

│ ├── index.txt ← Staging area

│ ├── commits/ ← Commit metadata storage

│ ├── objects/ ← File snapshot blobs

│ └── branches/ ← Branch references