Mini Git - A Version Control System

Project Report

Name: Modapu Adhivishnu
Roll Number: 2024202028

---

1. Introduction

Mini Git is a simplified version control system developed in C++ that replicates the essential functionality of Git. This project serves as an educational implementation to understand the internal workings of version control systems, including file hashing, object storage, data compression, and commit tracking.

The system implements 13 core Git commands that enable users to initialize repositories, track file changes, create commits, and navigate through project history. Built using modern C++ standards, the project demonstrates practical application of data structures, algorithms, cryptographic hashing (SHA-1), and file system operations.

Key learning objectives achieved through this implementation include:

- Understanding Git's internal object model (blobs, trees, commits)
- Implementing cryptographic hashing for content integrity
- Working with zlib compression for efficient storage
- Managing staging areas and commit workflows
- Creating command-line interfaces for version control operations

---

2. Commands Implemented and Usage

2.1 Repository Management

`init` - Initialize Repository

Purpose: Creates a new Mini Git repository in the current directory

Usage:

```
./mygit init
```

Output:

```
.mygit Git directory created successfully
```

What it does:

- Creates `.mygit/` directory structure
- Initializes `objects/`, `refs/heads/`, `logs/` folders
- Creates empty `index` and `HEAD` files
- Sets up the foundation for version control

`status` - Check Repository Status

Purpose: Shows the current state of files in the working directory

Usage:

```
./mygit status
```

Sample Output:

```
HEAD commit: a1b2c3d4...

Changes to be committed:
        new file:   file1.txt
        modified:   file2.txt

Changes not staged for commit:
        modified:   file3.txt

Untracked files:
        newfile.txt
```

What it shows:

- Files ready to be committed (staged)
- Modified files not yet staged
- New files not being tracked

2.2 File Management

`add` - Stage Files for Commit

Purpose: Adds files to the staging area

Usage:

```
./mygit add filename.txt        ->Add single file
./mygit add folder/             ->Add entire folder
./mygit add .                   ->Add all files
```

Output:

```
Added to staging area: filename.txt
SHA-1: a1b2c3d4e5f6...
Blob object written to: .mygit/objects/a1/b2c3d4e5f6...
```

What it does:

- Calculates SHA-1 hash of file content
- Creates blob object in `.mygit/objects/`
- Updates index with file information

`hash-object` - Create Object Hash

Purpose: Computes SHA-1 hash of a file

Usage:

```
./mygit hash-object filename.txt      # Just show hash
./mygit hash-object -w filename.txt  # Store in repository
```

Output:

```
SHA-1: a1b2c3d4e5f6789012345678901234567890abcd
Blob object written to: .mygit/objects/a1/b2c3d4e5f6789...
```

`cat-file` - View Object Contents

Purpose: Displays information about repository objects

Usage:

```
./mygit cat-file -p a1b2c3d4...  -> Show content
./mygit cat-file -t a1b2c3d4...  -> Show type
./mygit cat-file -s a1b2c3d4...  -> Show size
```

Sample Output:

```
$ -p flag (print content):
Hello, World!
This is file content.

$ -t flag (show type):
blob
```

$ -s flag (show size):

32

2.3 Commit Operations

`commit` - Create Commit

Purpose: Creates a permanent snapshot of staged changes

Usage:

```
./mygit commit -m "Your commit message"
```

Output:

```
f7e8d9c2b1a098765432109876543210fedcba09
```

What it does:

- Creates tree object from staged files
- Generates commit object with metadata
- Updates HEAD to point to new commit
- Clears staging area

`log` - View Commit History

Purpose: Displays commit history

Usage:

```
./mygit log
```

Sample Output:

```
Commit: f7e8d9c2b1a098765432109876543210fedcba09
```

```
Parent: a1b2c3d4e5f6789012345678901234567890abcd
Committer: Author <author@example.com>
Date: 1703001234 +0000
Message: Add new feature

Commit: a1b2c3d4e5f6789012345678901234567890abcd
Committer: Author <author@example.com>
Date: 1703000234 +0000
Message: Initial commit
```

2.4 Tree Operations

`write-tree` - Create Tree Object

Purpose: Creates tree object from current directory

Usage:

```
./mygit write-tree
```

Output:

```
Creating tree structure for: "/current/path"
Created tree object with hash: b3c4d5e6f7890123456789012345678901234abc
b3c4d5e6f7890123456789012345678901234abc
```

`ls-tree` - List Tree Contents

Purpose: Shows contents of a tree object

Usage:

```
./mygit ls-tree b3c4d5e6f789...          # Detailed view
./mygit ls-tree --name-only b3c4d5e6...  # Names only
```

Sample Output:

```
# Detailed view:
100644 blob a1b2c3d4e5f6789...    file1.txt
100644 blob f7e8d9c2b1a09876...    file2.txt
040000 tree b3c4d5e6f7890123...    subfolder

# Name only:
file1.txt
file2.txt
subfolder
```

`show` - Display Commit Details

Purpose: Shows commit information and changes

Usage:

```
./mygit show                    -> Show HEAD commit
./mygit show f7e8d9c2b1a0...   ->Show specific commit
```

Sample Output:

```
commit f7e8d9c2b1a098765432109876543210fedcba09
Author: Author <author@example.com>
Date: Committer <committer@example.com> 1703001234 +0000

    Add new feature
```

```
diff --git a/newfile.txt b/newfile.txt
new file mode 100644
index 0000000..a1b2c3d
--- /dev/null
+++ b/newfile.txt
+Hello, World!
+This is new content.
```

2.5 Navigation and Reset

`checkout` - Switch to Commit

Purpose: Restores working directory to a specific commit

Usage:

```
./mygit checkout f7e8d9c2b1a0987654321098765432...
```

Output:

```
Checking out commit f7e8d9c2b1a098765432109876543210fedcba09
Tree SHA: b3c4d5e6f7890123456789012345678901234abc
Clearing working directory...
Restoring files from tree...
Restored file: file1.txt
Restored file: file2.txt
Created directory: subfolder
Successfully checked out commit f7e8d9c2b1a0987...
```

Important Warning: This overwrites current files. Consider backing up your work first.

Recommendation: create another folder,copy “.mygit” and the executable file “mygit” to the folder you created

---

`reset` - Undo Changes

Purpose: Various reset operations to undo changes

Usage:

```
./mygit reset                          # Unstage all files
./mygit reset filename.txt             # Unstage specific file
./mygit reset --hard f7e8d9c2b1a0...   # Reset to commit
```

Sample Outputs:

```
# Basic reset:
Index cleared successfully

# File-specific reset:
Unstaged 'filename.txt'

# Hard reset:
Resetting to commit f7e8d9c2b1a0987...
HEAD is now at f7e8d9c2
```

---

3. Complete Workflow Example

Here's a typical workflow demonstrating how to use Mini Git:

1. Initialize repository

```
./mygit init
```

2. Create some files

```
echo "Hello World" > hello.txt
echo "First program" > program.cpp
```

3. Check status

```
./mygit status
```

4. Add files to staging

```
./mygit add hello.txt
./mygit add program.cpp
```

5. Check status again

```
./mygit status
```

6. Create first commit

```
./mygit commit -m "Initial commit with hello.txt and program.cpp"
```

7. View commit history

```
./mygit log
```

8. Modify a file

```
echo "Hello World Updated" > hello.txt
```

9. Check what changed

```
./mygit status
```

10. Add and commit changes

```
./mygit add hello.txt
./mygit commit -m "Update hello.txt content"
```

11. View detailed commit info

```
./mygit show
```

12. Go back to previous commit

```
./mygit checkout [previous-commit-hash]
```

---

4. Build and Setup Instructions

Prerequisites:

- C++17 compatible compiler (g++ recommended)
- OpenSSL development libraries
- zlib development libraries

Build Command:

```
g++ -std=c++17 *.cpp -o mygit -lssl -lcrypto -lz
```