***Diff + CLI + Demo***

***Diff Command***

• Compares two commits line-by-line. The `diff` function in MiniGit compares the contents of two commits (or a commit and the staging area).

• Detects differences between versions of files.

• Uses SHA-1 hashes to identify file versions.

• Output style similar to Git:

- '+' means added

- '-' means removed

- ' ' means unchanged

Key steps from the code:

1. Resolve commit hashes from branch names if needed:

- If `c1` or `c2` is a branch, read its commit hash from `.minigit/refs/<branch>`

2. Extract tree hashes from commit objects:

- Commit objects are stored in `.minigit/objects/`.  
 - The code reads lines starting with `tree: ` to find the tree hash.

3. Recursively collect files using the `collect\_files()` function:

- This function traverses tree objects recursively.  
 - Each file path and its blob SHA-1 hash are collected in a map.

4. Line-by-line comparison:

- Unique file paths from both commits are collected.  
 - For each file, lines from both versions are compared.

- Output follows Git-like syntax:  
 - `+` for additions  
 - `-` for deletions  
 - (space) for unchanged lines

This implementation captures basic textual differences between files and presents them clearly in the console.

void diff(const std::string& c1, const std::string& c2 = "") {

// Get commit hashes from branch names if provided

std::string commit1 = c1;

if (fs::exists(refs\_dir + "/" + c1)) commit1 = read\_file(refs\_dir + "/" + c1);

std::string commit2 = c2;

if (!c2.empty() && fs::exists(refs\_dir + "/" + c2)) commit2 = read\_file(refs\_dir + "/" + c2);

// Load file trees for both commits

std::unordered\_map<std::string, std::string> files1, files2;

// Load first commit's files

std::string tree1;

std::string content = read\_file(objects\_dir + "/" + commit1);

std::stringstream ss(content);

std::string line;

while (std::getline(ss, line)) {

if (line.find("tree: ") == 0) {

tree1 = line.substr(6);

break;

}

}

collect\_files(tree1, "", files1);

// Load second commit's files (or working directory if no second commit)

if (c2.empty()) {

for (auto& [path, hash] : staging\_area) files2[path] = hash;

} else {

std::string tree2;

content = read\_file(objects\_dir + "/" + commit2);

ss.clear();

ss.str(content);

while (std::getline(ss, line)) {

if (line.find("tree: ") == 0) {

tree2 = line.substr(6);

break;

}

}

collect\_files(tree2, "", files2);

}

// Compare all files

std::unordered\_set<std::string> all\_paths;

for (auto& [p, \_] : files1) all\_paths.insert(p);

for (auto& [p, \_] : files2) all\_paths.insert(p);

for (auto& path : all\_paths) {

std::string h1 = files1.count(path) ? files1[path] : "";

std::string h2 = files2.count(path) ? files2[path] : "";

if (h1 == h2) continue;

// Read file contents

std::vector<std::string> lines1, lines2;

if (!h1.empty()) {

std::string content = read\_file(objects\_dir + "/" + h1);

std::stringstream ss(content);

std::string line;

while (std::getline(ss, line)) lines1.push\_back(line);

}

if (!h2.empty()) {

std::string content = c2.empty() ? read\_file(path) : read\_file(objects\_dir + "/" + h2);

std::stringstream ss(content);

std::string line;

while (std::getline(ss, line)) lines2.push\_back(line);

}

// Print diff header

std::cout << "--- " << path << "\n+++ " << path << "\n";

// Print line-by-line differences

for (size\_t i = 0, j = 0; i < lines1.size() || j < lines2.size();) {

if (i < lines1.size() && j < lines2.size() && lines1[i] == lines2[j]) {

std::cout << " " << lines1[i] << "\n";

++i; ++j;

} else if (j < lines2.size() && (i >= lines1.size() || lines1[i] != lines2[j])) {

std::cout << "+" << lines2[j] << "\n";

++j;

} else {

std::cout << "-" << lines1[i] << "\n";

++i;

}

}

}

}

**Command-Line Interface (CLI)**

MiniGit accepts commands via the terminal. These are handled in the `main()` function at the bottom of the code.

**.how to use MiniGit from the command line:**

1. **Initialize a new repository: ./minigit init**
2. **Add files to staging area**: ./minigit add file.txt

./minigit add directory/

1. **Commit changes**: ./minigit commit -m "Initial commit"
2. **View commit history**: ./minigit log
3. **Create a new branch: ./minigit branch new-feature**
4. **Switch branches**: ./minigit checkout new-feature
5. **Merge branches**: ./minigit merge main
6. **View differences**: ./minigit diff HEAD~1 HEAD # Compare last two commits

./minigit diff branch1 branch2 # Compare two branches

./minigit diff HEAD # Compare HEAD with working directory

The CLI serves as the user interface to control MiniGit's core functionality through intuitive commands.

**Demo Steps – From Init to Diff**

To MiniGit’s capabilities:

1. Run `minigit init` to set up the repository.

2. Create a file (e.g., `hello.txt`) and run `minigit add hello.txt`.

3. Commit with `minigit commit -m "initial commit"`.

4. Modify `hello.txt`, add again, and commit with another message.

5. Run `minigit diff <first\_commit\_hash> <second\_commit\_hash>` to display changes.

**Demo Screenshots**

1. **Initialization and First Commit**:

$ ./minigit init

Initialized MiniGit repo

$ echo "Hello World" > hello.txt

$ ./minigit add hello.txt

Added hello.txt

$ ./minigit commit -m "First commit"

[MiniGit] Commit successful: a94a8fe5cc...

1. **Viewing History**:

$ ./minigit log

commit a94a8fe5cc...

Date: 2025-06-20 14:30:45

First commit

1. **Creating and Switching Branches**:

$ ./minigit branch feature

Branch feature created

$ ./minigit checkout feature

Checked out feature

1. **Viewing Differences**:

$ echo "New line" >> hello.txt

$ ./minigit diff HEAD

--- hello.txt

+++ hello.txt

Hello World

+New line

**Summary**

• `diff` helps visualize changes in project history.

• CLI allows complete control over MiniGit.

• Demo shows real-world usage.