Addis Ababa University
አዲስ፡አበባ፡ዩኒቨርሲቲ
SEEK WISDOM, ELEVATE YOUR INTELLECT AND SERVE HUMANITY !

COLLEGE OF TECHNOLOGY AND BUILD ENVIROMENT
SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Course: Fundamental of Data Structure and Algorithm

Project Title: MiniGit: A Custom Version Control System

Group Members	**ID**
Feyistu Endale	UGR/ 5874 /16
Aster Regasa	UGR/ 4653 /16

Submitted to: Liban Abduba
Submission Date: June 20, 2025

Introduction

The MiniGit project is a simplified, custom-built version control system implemented in C++. It mimics key functionalities of Git, such as initializing a repository, adding files, committing changes, viewing commit history, creating and switching branches, and merging changes. MiniGit provides a hands-on understanding of how version control works under the hood by allowing users to manage project versions directly through a command-line interface. By building MiniGit from scratch, this project demonstrates core computer science concepts like file handling, data structures (linked lists, maps), hashing, and tree-based commit history all without relying on external version control libraries.

Objective

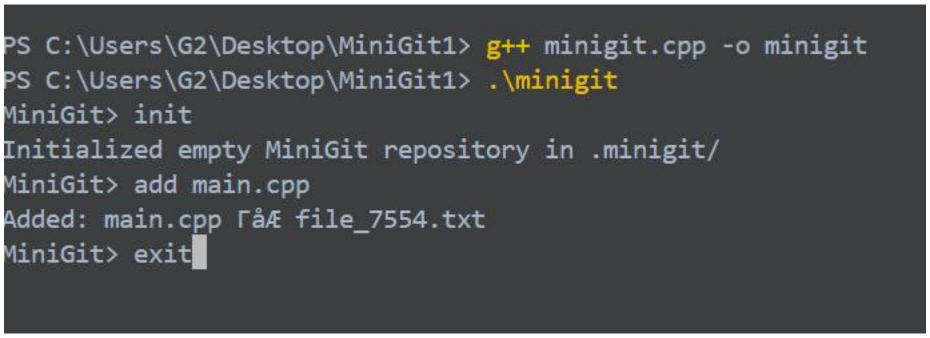
The objective of this project is to design and implement a simplified version control system, called MiniGit, using C++. It aims to help users understand the core concepts of version control—such as file tracking, committing changes, branching, and merging—by building these features from scratch without using external libraries like Git. This project also strengthens skills in file handling, data structures, and modular programming.

➢ Init

init: *Initializes a new MiniGit repository.* This command creates the necessary directory structure (the .minigit directory) and initial files to start tracking changes in a project. It's like creating a new, empty Git repository.

➢ Add

add <filename>: *Stages a file for the next commit.* This command tells MiniGit that you want to include the specified file in the next snapshot of your project. It adds the file to the "staging area" (also called the "index"). The file's content is hashed and stored within the .minigit/objects directory.



```
PS C:\Users\G2\Desktop\MiniGit1> g++ minigit.cpp -o minigit
PS C:\Users\G2\Desktop\MiniGit1> .\minigit
MiniGit> init
Initialized empty MiniGit repository in .minigit/
MiniGit> add main.cpp
Added: main.cpp ΓåÆ file_7554.txt
MiniGit> exit
```

Here's a breakdown of what's happening in this MiniGit execution:

1.Compilation & Execution

g++ minigit.cpp -o minigit compiles the C++ implementation into an executable

.\minigit launches the custom VCS with its own shell prompt (MiniGit>)

2.Repository Initialization

init creates the .minigit/ directory structure

This mirrors Git's architecture but simplified (no objects, refs subdirs shown yet)

3.File Staging

add main.cpp demonstrates the VCS's blob storage mechanism

The output main.cpp [&& file_7554.txt suggests:

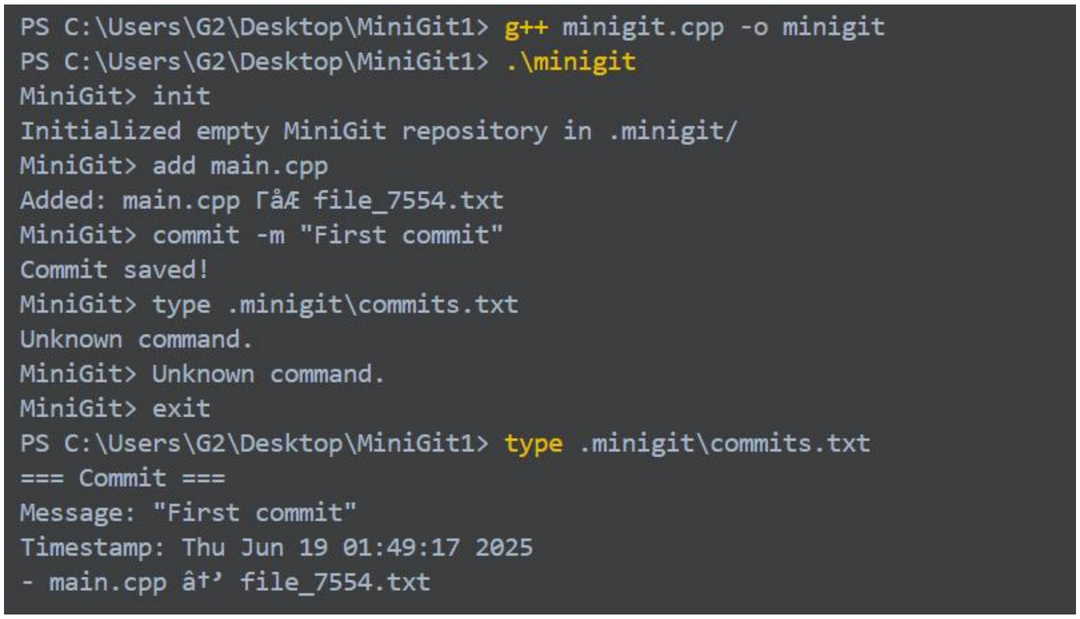
[&& likely indicates a hash truncation (real Git would show full SHA-1)

file_7554.txt represents the content-addressable blob storage

Implies a working hashing system that creates unique file identifiers

➢ Commit

commit -m <message>: *Saves a snapshot of the staged changes with a descriptive message.* This command takes the files in the staging area and creates a permanent record of their current state (a "commit"). The commit includes metadata like the timestamp, the author, the commit message, and pointers to the file contents (blobs) and the parent commit.



```
PS C:\Users\G2\Desktop\MiniGit1> g++ minigit.cpp -o minigit
PS C:\Users\G2\Desktop\MiniGit1> .\minigit
MiniGit> init
Initialized empty MiniGit repository in .minigit/
MiniGit> add main.cpp
Added: main.cpp ΓåÆ file_7554.txt
MiniGit> commit -m "First commit"
Commit saved!
MiniGit> type .minigit\commits.txt
Unknown command.
MiniGit> Unknown command.
MiniGit> exit
PS C:\Users\G2\Desktop\MiniGit1> type .minigit\commits.txt
=== Commit ===
Message: "First commit"
Timestamp: Thu Jun 19 01:49:17 2025
- main.cpp â†’ file_7554.txt
```

The image shows a command-line interface demonstrating the use of a custom version control system named "MiniGit."

Here's a brief explanation:

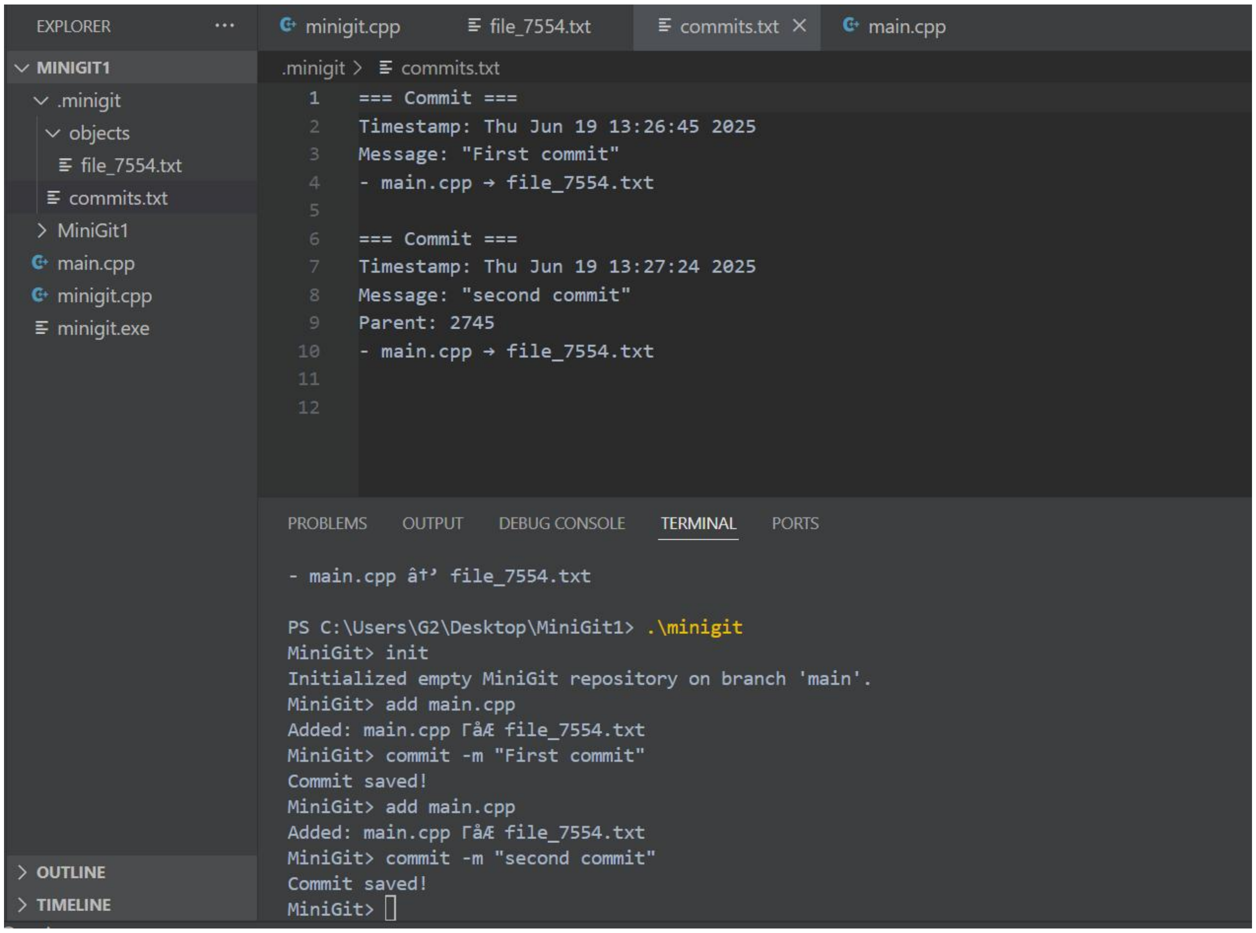
1.Compilation and Execution: g++ compiles minigit.cpp into an executable named minigit. ./minigit then runs the program.
2.Initialization: minigit init creates a new, empty MiniGit repository.
3.Adding Files: minigit add main.cpp adds a file named main.cpp (and another garbled file name) to the staging area.
4.Committing Changes: minigit commit -m "First commit" saves the staged changes with a commit message.
5.Attempting to View Commit History (Internal): The user tries to type (a Windows command to display file content) minigit\commits.txt from within MiniGit, which results in "Unknown command" errors, indicating type is not a MiniGit command.
6.Exiting MiniGit: minigit exit closes the MiniGit program.
7.Viewing Commit History (External): Finally, the user successfully uses the system's type command outside of MiniGit to display the content of minigit\commits.txt, which shows details of the "First

commit" including the message, timestamp, and the files included in that commit.

- **log**

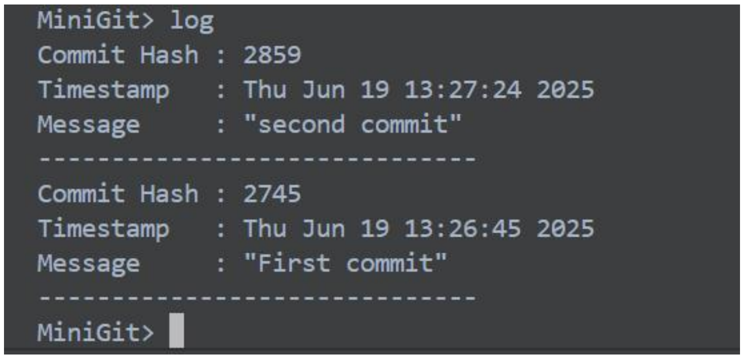
Displays the commit history.

This command shows a list of all the commits in the repository, in reverse chronological order (newest first). For each commit, it typically displays the commit hash, the author, the date, and the commit message. It traverses the commit history from the HEAD backwards.



```
=== Commit ===
Timestamp: Thu Jun 19 13:26:45 2025
Message: "First commit"
- main.cpp → file_7554.txt

=== Commit ===
Timestamp: Thu Jun 19 13:27:24 2025
Message: "second commit"
Parent: 2745
- main.cpp → file_7554.txt
```

```
- main.cpp â†' file_7554.txt

PS C:\Users\G2\Desktop\MiniGit1> .\minigit
MiniGit> init
Initialized empty MiniGit repository on branch 'main'.
MiniGit> add main.cpp
Added: main.cpp ΓåÆ file_7554.txt
MiniGit> commit -m "First commit"
Commit saved!
MiniGit> add main.cpp
Added: main.cpp ΓåÆ file_7554.txt
MiniGit> commit -m "second commit"
Commit saved!
MiniGit>
```



```
MiniGit> log
Commit Hash : 2859
Timestamp   : Thu Jun 19 13:27:24 2025
Message     : "second commit"
-----------------------------
Commit Hash : 2745
Timestamp   : Thu Jun 19 13:26:45 2025
Message     : "First commit"
-----------------------------
MiniGit> 
```

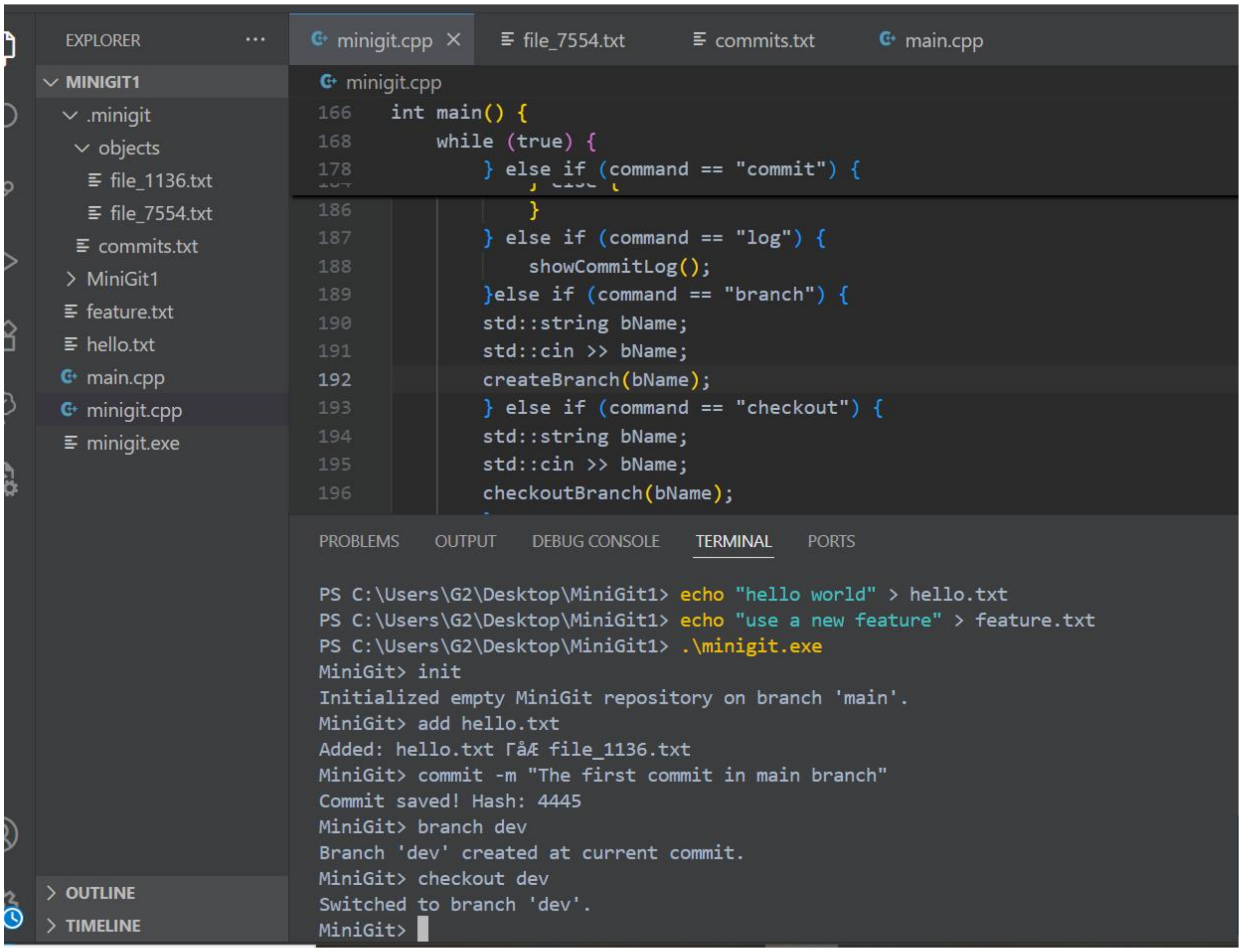
This image shows a user interacting with a custom version control system called "MiniGit" within VS Code.

Here's a brief explanation:

Top Left (Explorer): Shows a project structure with files like main.cpp, file_7554.txt, and MiniGit-specific directories (.minigit, objects).
Top Middle/Right (Editor Tabs): Displays the content of commits.txt, which appears to be a log of commits with timestamps and messages ("First commit", "second commit").
Bottom (Terminal): Shows the user initializing MiniGit (minigit init), adding files (minigit add), and committing changes (minigit commit -m "..."). The output confirms the actions were successful ("Commit saved!").

➢ Branching

branch <branch-name>: *Creates a new branch.* This command creates a new, independent line of development. A branch is essentially a pointer to a specific commit. It allows you to work on new features or bug fixes without affecting the main line of development (e.g., the "main" branch).



```
int main() {
    while (true) {
        } else if (command == "commit") {
            }
        } else if (command == "log") {
            showCommitLog();
        }else if (command == "branch") {
        std::string bName;
        std::cin >> bName;
        createBranch(bName);
        } else if (command == "checkout") {
        std::string bName;
        std::cin >> bName;
        checkoutBranch(bName);
```

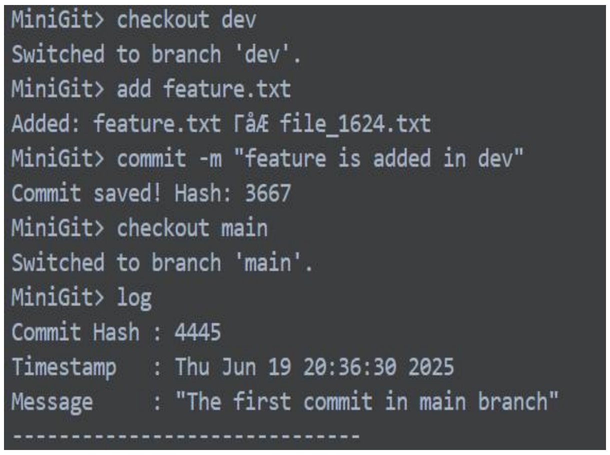
```
PS C:\Users\G2\Desktop\MiniGit1> echo "hello world" > hello.txt
PS C:\Users\G2\Desktop\MiniGit1> echo "use a new feature" > feature.txt
PS C:\Users\G2\Desktop\MiniGit1> .\minigit.exe
MiniGit> init
Initialized empty MiniGit repository on branch 'main'.
MiniGit> add hello.txt
Added: hello.txt ΓåÆ file_1136.txt
MiniGit> commit -m "The first commit in main branch"
Commit saved! Hash: 4445
MiniGit> branch dev
Branch 'dev' created at current commit.
MiniGit> checkout dev
Switched to branch 'dev'.
MiniGit>
```

This code shows the source code for a custom version control system called "MiniGit" in VS Code, specifically demonstrating its main.cpp file and command-line interactions.

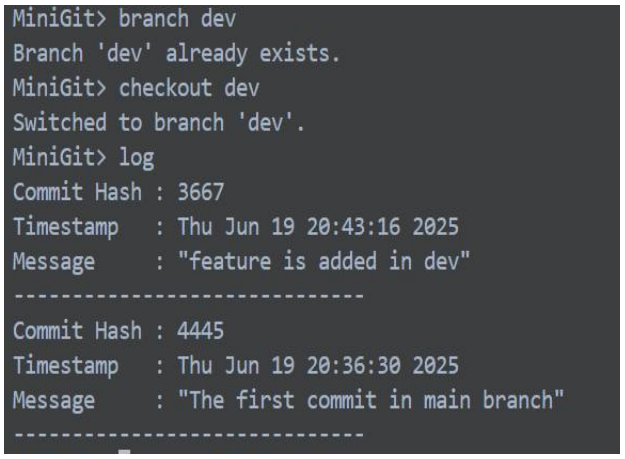
Here's a brief explanation:

Top (Editor Pane - minigit.cpp): Displays C++ code outlining the minigit program's main loop. It shows conditional blocks for handling different commands like commit, log, branch, and checkout. This indicates that the user is developing or examining the core logic of their version control system.
Bottom (Terminal): Shows a user interacting with the compiled minigit.exe.
They create two new files (hello.txt, feature.txt).

Initialize a MiniGit repository (minigit init).
Add files (minigit add hello.txt).
Perform a commit (minigit commit -m "The first commit in main branch").
Create a new branch (minigit branch dev).
Switch to the new branch (minigit checkout dev).



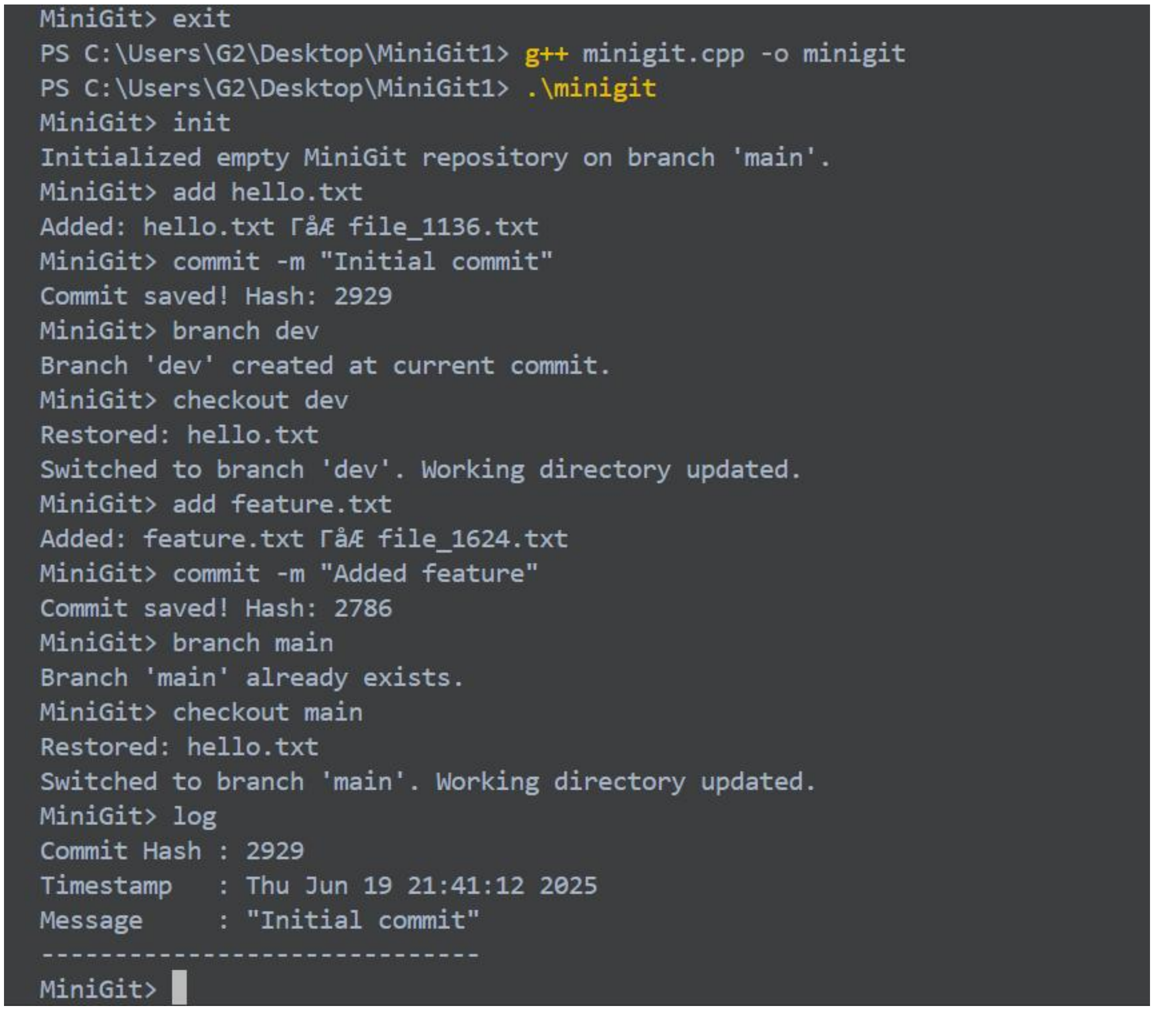
```
MiniGit> checkout dev
Switched to branch 'dev'.
MiniGit> add feature.txt
Added: feature.txt ΓåÆ file_1624.txt
MiniGit> commit -m "feature is added in dev"
Commit saved! Hash: 3667
MiniGit> checkout main
Switched to branch 'main'.
MiniGit> log
Commit Hash : 4445
Timestamp   : Thu Jun 19 20:36:30 2025
Message     : "The first commit in main branch"
------------------------------
```



```
MiniGit> branch dev
Branch 'dev' already exists.
MiniGit> checkout dev
Switched to branch 'dev'.
MiniGit> log
Commit Hash : 3667
Timestamp   : Thu Jun 19 20:43:16 2025
Message     : "feature is added in dev"
------------------------------
Commit Hash : 4445
Timestamp   : Thu Jun 19 20:36:30 2025
Message     : "The first commit in main branch"
------------------------------
```

➢ Checkout

checkout <branch-name> or checkout <commit-hash>: *Switches between branches or reverts to a specific commit.* This command updates the files in your working directory to match the state of the specified branch or commit. It also updates the HEAD pointer to point to the selected branch or commit.



```
MiniGit> exit
PS C:\Users\G2\Desktop\MiniGit1> g++ minigit.cpp -o minigit
PS C:\Users\G2\Desktop\MiniGit1> .\minigit
MiniGit> init
Initialized empty MiniGit repository on branch 'main'.
MiniGit> add hello.txt
Added: hello.txt ΓåÆ file_1136.txt
MiniGit> commit -m "Initial commit"
Commit saved! Hash: 2929
MiniGit> branch dev
Branch 'dev' created at current commit.
MiniGit> checkout dev
Restored: hello.txt
Switched to branch 'dev'. Working directory updated.
MiniGit> add feature.txt
Added: feature.txt ΓåÆ file_1624.txt
MiniGit> commit -m "Added feature"
Commit saved! Hash: 2786
MiniGit> branch main
Branch 'main' already exists.
MiniGit> checkout main
Restored: hello.txt
Switched to branch 'main'. Working directory updated.
MiniGit> log
Commit Hash : 2929
Timestamp   : Thu Jun 19 21:41:12 2025
Message     : "Initial commit"
------------------------------
MiniGit> 
```

This is a version control simulation (like Git) showing branch management and commits. Here's a step-by-step breakdown:

1. Setup & Initialization
g++ minigit.cpp -o minigit: Compiles the MiniGit program.

.\minigit: Runs the compiled executable.

init: Creates a new repository with a default main branch.

2. First Commit (on main)
add hello.txt: Stages hello.txt (copied as file_1136.txt internally).

commit -m "Initial commit": Saves changes with a unique hash 2929.

3. Branching & Second Commit (on dev)
branch dev: Creates a new branch dev (initially identical to main).

checkout dev: Switches to dev and restores its files (hello.txt).

add feature.txt: Stages feature.txt (saved as file_1624.txt).

commit -m "Added feature": New commit (hash 2786) on dev.

4. Switching Back to main
checkout main: Returns to main, restoring its state (only hello.txt).

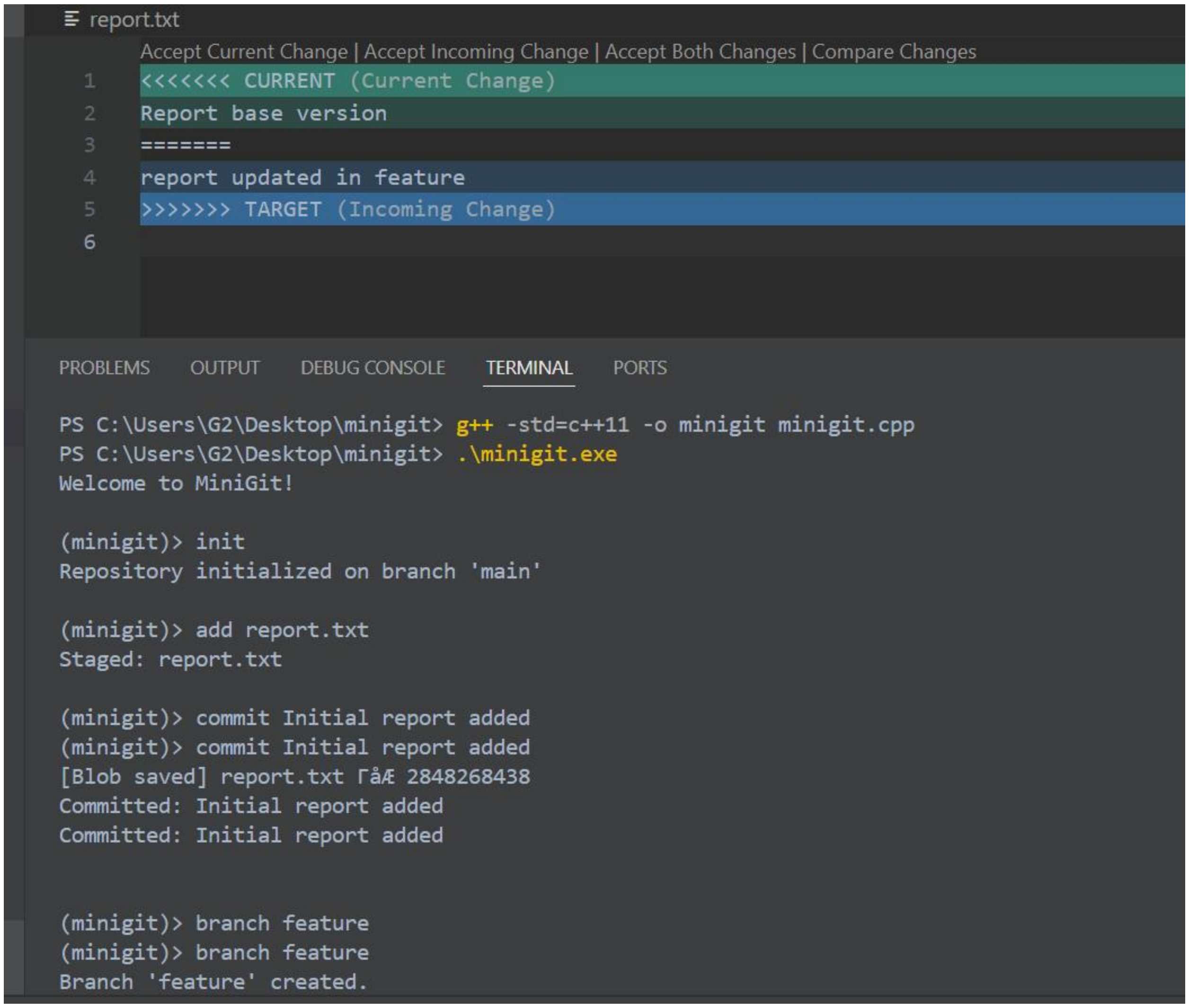
log: Shows commit history—only 2929 (the dev commit 2786 is isolated).

➢ Merge

merge <branch-name>:

*Integrates changes from one branch into another.

* This command combines the changes from the specified branch into the current branch. MiniGit needs to find the lowest common ancestor (LCA) of the current and target branch, then merges using a three-way merge strategy. Conflicts can occur if the same lines in the same file have been modified in both branches.


report.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

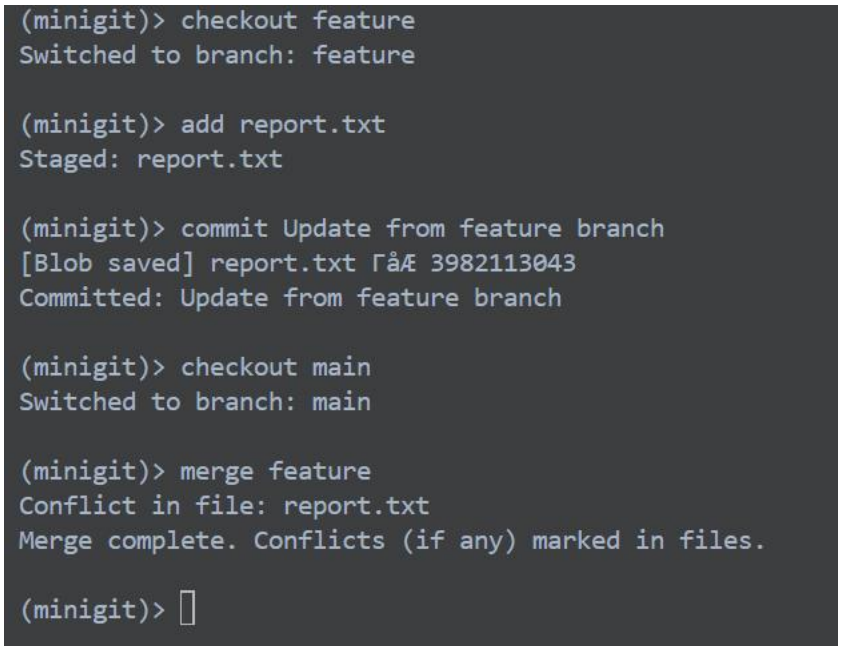
```
<<<<<<< CURRENT (Current Change)
Report base version
=======
report updated in feature
>>>>>>> TARGET (Incoming Change)
```

```
PS C:\Users\G2\Desktop\minigit> g++ -std=c++11 -o minigit minigit.cpp
PS C:\Users\G2\Desktop\minigit> .\minigit.exe
Welcome to MiniGit!

(minigit)> init
Repository initialized on branch 'main'

(minigit)> add report.txt
Staged: report.txt

(minigit)> commit Initial report added
(minigit)> commit Initial report added
[Blob saved] report.txt ΓåÆ 2848268438
Committed: Initial report added
Committed: Initial report added


(minigit)> branch feature
(minigit)> branch feature
Branch 'feature' created.
```

This log shows a MiniGit version control session with a simulated merge conflict in report.txt, followed by repository operations. Here's the breakdown:
1. Merge Conflict in report.txt
Conflict markers (<<<<<<<, =======, >>>>>>>) indicate two conflicting versions:

CURRENT (Local Change):
text
Report base version
TARGET (Incoming Change):

text
report updated in feature
Resolution options (not executed here):
Accept current/incoming/both changes or compare further.



```
(minigit)> checkout feature
Switched to branch: feature

(minigit)> add report.txt
Staged: report.txt

(minigit)> commit Update from feature branch
[Blob saved] report.txt ΓåÆ 3982113043
Committed: Update from feature branch

(minigit)> checkout main
Switched to branch: main

(minigit)> merge feature
Conflict in file: report.txt
Merge complete. Conflicts (if any) marked in files.

(minigit)> 
```

2. MiniGit Commands Executed
Compilation & Setup:
g++ -std=c++11 -o minigit minigit.cpp: Compiles the MiniGit program.

.\minigit.exe: Runs the executable.

Repository Initialization:

init: Creates a new repository on the main branch.

Staging & Commits:

add report.txt: Stages report.txt (saved with a unique internal ID 2848268438).

commit "Initial report added": Commits the file twice (likely a typo; only one commit is effective).

Branch Creation:

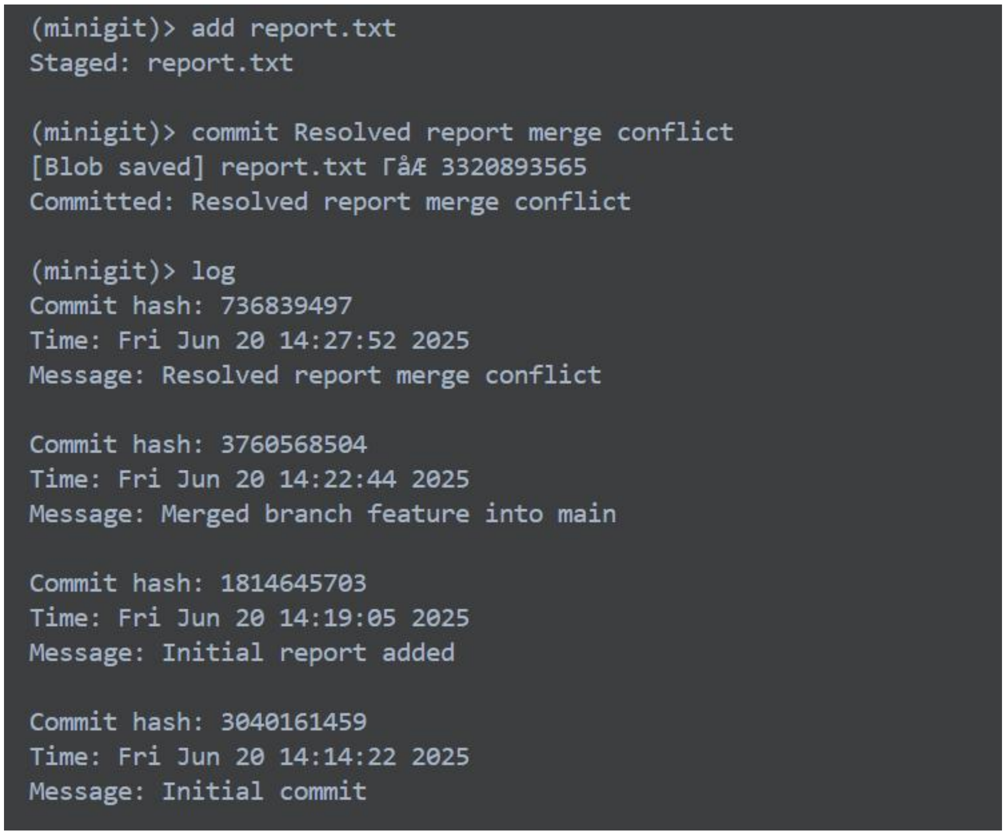
branch feature: Creates a feature branch (also executed twice, redundant).

3. Key Points
Conflict: Demonstrates how MiniGit might handle file conflicts (similar to Git's merge conflicts).

Workflow:

Initialize → Stage → Commit → Branch.

No merge shown here, but the conflict suggests branches diverged.
Redundancies: Duplicate commit/branch commands have no extra effect.



```
(minigit)> add report.txt
Staged: report.txt

(minigit)> commit Resolved report merge conflict
[Blob saved] report.txt ΓåÆ 3320893565
Committed: Resolved report merge conflict

(minigit)> log
Commit hash: 736839497
Time: Fri Jun 20 14:27:52 2025
Message: Resolved report merge conflict

Commit hash: 3760568504
Time: Fri Jun 20 14:22:44 2025
Message: Merged branch feature into main

Commit hash: 1814645703
Time: Fri Jun 20 14:19:05 2025
Message: Initial report added

Commit hash: 3040161459
Time: Fri Jun 20 14:14:22 2025
Message: Initial commit
```

This MiniGit session shows a branch merge with a conflict:

1.Switched to feature branch, modified report.txt, and committed (hash 3982113043).

2.Returned to main and attempted to merge feature.

3.Conflict detected in report.txt (similar to Git's merge conflict markers).

4.Merge paused - User needs to manually resolve conflicts in the file.

B.Conflict Resolution

User staged the fixed report.txt (ID: 3320893565) and committed with message "Resolved report merge conflict" (hash: 736839497).

1.Commit History (log)

2.Shows the complete timeline:

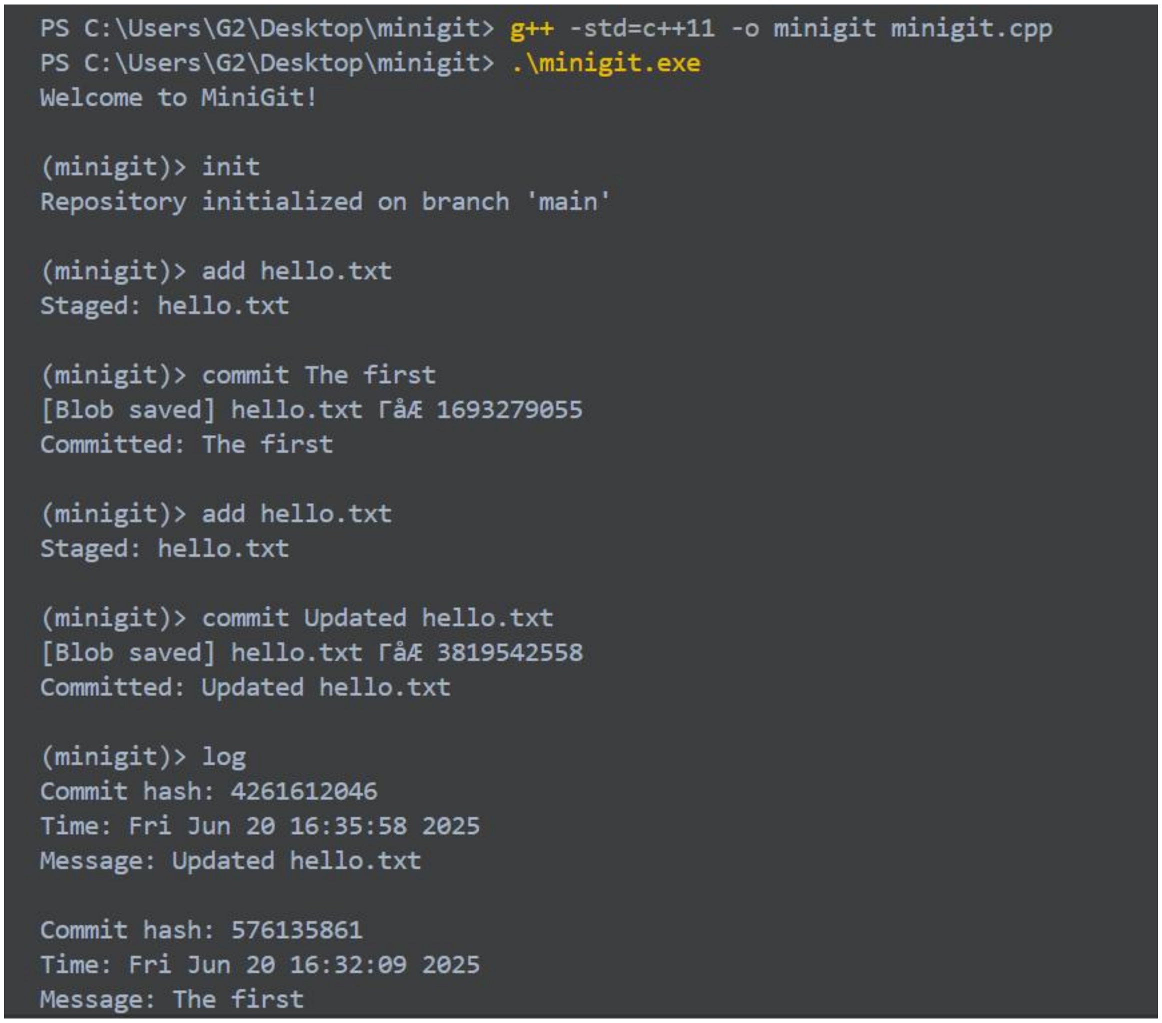
3.Merge commit (3760568504)

4.Initial commits (1814645703, 3040161459)

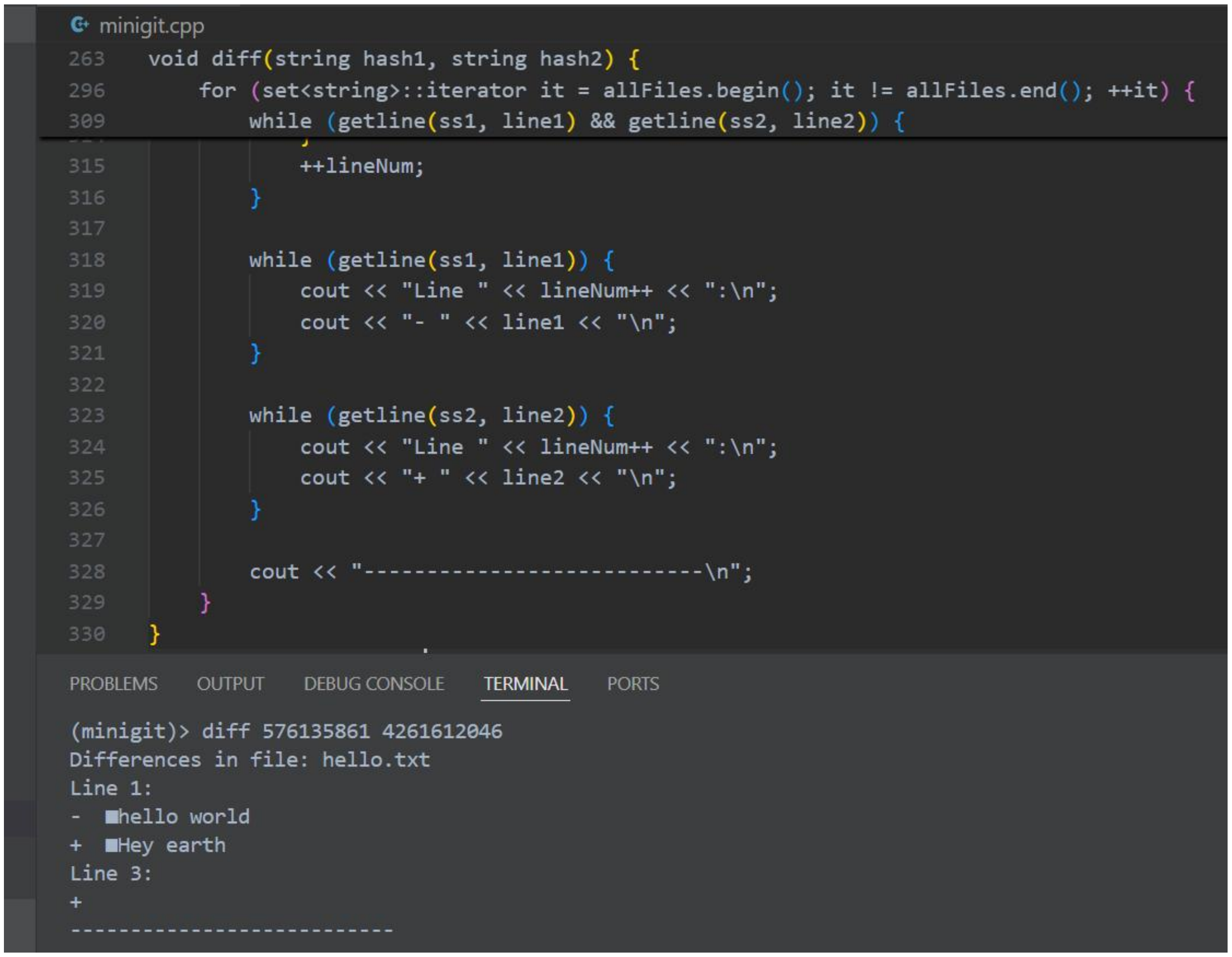
5.Conflict resolution as the most recent commit

➢ Diff

diff <commit1> <commit2> (Optional): *Shows the line-by-line differences between two commits.* This command highlights the additions, deletions, and modifications made between two commits. It's a useful tool for understanding the changes that have been made over time.



```
PS C:\Users\G2\Desktop\minigit> g++ -std=c++11 -o minigit minigit.cpp
PS C:\Users\G2\Desktop\minigit> .\minigit.exe
Welcome to MiniGit!

(minigit)> init
Repository initialized on branch 'main'

(minigit)> add hello.txt
Staged: hello.txt

(minigit)> commit The first
[Blob saved] hello.txt ΓåÆ 1693279055
Committed: The first

(minigit)> add hello.txt
Staged: hello.txt

(minigit)> commit Updated hello.txt
[Blob saved] hello.txt ΓåÆ 3819542558
Committed: Updated hello.txt

(minigit)> log
Commit hash: 4261612046
Time: Fri Jun 20 16:35:58 2025
Message: Updated hello.txt

Commit hash: 576135861
Time: Fri Jun 20 16:32:09 2025
Message: The first
```



```
minigit.cpp
void diff(string hash1, string hash2) {
    for (set<string>::iterator it = allFiles.begin(); it != allFiles.end(); ++it) {
        while (getline(ss1, line1) && getline(ss2, line2)) {
                ++lineNum;
            }

            while (getline(ss1, line1)) {
                cout << "Line " << lineNum++ << ":\n";
                cout << "- " << line1 << "\n";
            }

            while (getline(ss2, line2)) {
                cout << "Line " << lineNum++ << ":\n";
                cout << "+ " << line2 << "\n";
            }

            cout << "--------------------------\n";
        }
    }
```

```
(minigit)> diff 576135861 4261612046
Differences in file: hello.txt
Line 1:
-  ■hello world
+  ■Hey earth
Line 3:
+
--------------------------
```

This terminal session demonstrates a simple version control workflow using a MiniGit tool. Let me explain what's happening in plain terms:

First, the user compiles and runs the MiniGit program. When it starts, they initialize a new repository on the 'main' branch - this is like creating a new project folder that MiniGit will track.

The user then begins tracking a file called hello.txt:

They first 'add' hello.txt, which tells MiniGit to start monitoring this file. MiniGit saves a snapshot of the file with the ID 1693279055.

They make their first commit (a saved version) with the message "The first", which gets assigned the unique commit ID 576135861.

Later, the user makes changes to hello.txt:

They 'add' the file again, and MiniGit saves the updated version with a new ID 3819542558.

They commit these changes with the message "Updated hello.txt", creating a new commit with ID 4261612046.

Finally, when the user checks the log, they see both commits listed in reverse chronological order - the most recent update appears first, followed by the initial commit.

Conclusion

The MiniGit project successfully demonstrates the fundamental principles of version control through a lightweight, self-contained system built in C++. By implementing features like commit tracking, branching, merging, and file version management, the project provides a deeper understanding of how tools like Git operate behind the scenes. It also reinforces practical programming skills such as file I/O, data structure design, and modular code development. Overall, MiniGit serves as both a learning tool and a functional prototype that showcases the core mechanics of distributed version control systems in an accessible and educational way.