Time-Travelling File System

Long Assignment Report

Rami Jay Ketanbhai
Entry No: 2024CS10510

September 11, 2025

Contents

1 Introduction

This project implements a simplified *Time-Travelling File System* inspired by Git. Using self-implemented **heap, hashmap, lists** and other data structures.

2 Files Used

The code base is organised into the following files:

- **main.cpp** – Main files. Handles inputs and feeds to the filesystem.
- **File.hpp** – Defines the `File` class and involving the list of commands.
- **input.txt** - Allows user to give several inputs at once.
- **TreeNode.hpp** – Node structure for version trees.
- **HashMap.hpp** – Custom hash map with self-made hash functions for $O(1)$ lookups.
- **Heap.hpp** – Max-heap for analytics such as most recent files.
- **List.hpp** – Linked list used internally by the hash map.
- **compile.sh** – Bash script to compile and run on Linux/macOS.
- **run.ps1** – PowerShell script to compile and run on Windows.

3 How to Run

Compilation and execution can be automated with the scripts provided. It also takes user input to choose between automated input through input.txt or interactive input.

3.1 Linux / macOS : compile.sh

```
chmod +x compile.sh
./compile.sh
```

The first line is used to set bash file to make executable files. The script prompts the user to choose between running with ‘input.txt‘ or interactive input.

3.2 Windows : run.ps1

```
.\run.ps1
```

Before first use, enable script execution:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

The script asks the same question : use the input file or type commands interactively.

4 Commands

Each command is parsed in `main.cpp` using `istringstream`. Key commands include:

CREATE

`CREATE <filename>` Creates a new file with root version 0 which is already snapshotted. I have set initial message to INITIAL SNAPSHOT and empty content.

INSERT
`INSERT <filename> <content>` Appends content to existing file content. A new version is created if the active version is a snapshot.

UPDATE
`UPDATE <filename> <content>` Replaces content of existing file content. A new version is created if the active version is a snapshot.

SNAPSHOT
`SNAPSHOT <filename> <message>` Marks the active version as immutable with a message and timestamp.

ROLLBACK
`ROLLBACK <filename> [versionID]` Moves the active version pointer to a specific ID or, if omitted, to the parent version.

HISTORY
`HISTORY <filename>` Lists all snapshot versions along the path from the active node to the root. It shows all versions time-stamps and message of snapshot.

RECENT_FILES
`RECENT_FILES [NUM]` shows the most recently modified `NUM` files along with last modified time-stamp that shows how long ago file was created using the custom heap and hashtable.

BIGGEST_TREES
`BIGGEST_TREES [NUM]` shows the `NUM` files with largest number of versions along with total versions of that file using the custom heap and hashtable.

EXIT
I have defined this command in order to end the program.

5 Data Structures Used

The implementation relies on custom, from-scratch data structures:

- **Tree** (`TreeNode.hpp`) – Each node stores content, message, timestamps, version-id and links to parent and children nodes.
- **HashMap** (`HashMap.hpp`) – Provides average $O^*(1)$ insert, lookup, and delete operations. Used for mapping version IDs to tree nodes and filenames to file objects and indexing in heaps.
- **Heap** (`Heap.hpp`) – A max-heap storing pairs of filename and last-modified time or version count, used for `RECENT_FILES` and `BIGGEST_TREES`.
- **Linked List** (`List.hpp`) – Internal chaining structure for the hash maps(way around to collisions).

6 Error Handling

Error handling is implemented for inputs:

- In case a file with given name already exists, it shows user that **file already exist**
- There an upper-bound on number of files I can modify per second to 10e8 as a **tie-breaker** in case of same creation time.
- In case when user tries to access a **file not present** in `file_map`, it prints `"Filename not present in system.Do you want to add File?(Use CREATE)"`
- In case of `ROLLBACK`, if **invalid version_id** is given or is **not numeric**, it tells user to give correct version_id.
- Invalid commands print a clear message **Invalid Command**.
- Attempting to **snapshot** an already snapshotted version reports the existing snapshot time, telling the user to use Insert/Update to create a new version.
- In case of **not giving filename** to system, it asks user to enter filename and retry.
- In case of not giving NUM to **Recent_files or Biggest_Trees**, it says user "No index provided. Showing all files in recent order" or "No index provided. Showing all files in order of size"
- In case of empty **Recent_Files or Biggeest_Trees**, it tells user that no file is present in system.
- In case of non-integer inputs in **Recent_Files or Biggeest_Trees**, tells user to give integer input.
- In case of negaative input in **Recent_Files or Biggeest_Trees or Rollback**, tells user negative value is not permitted.