 # Version-Control File System – README

Overview

This project implements a **mini file system with version control**, supporting operations like file creation, updates, rollbacks, snapshots, and queries for most recent/most versions files.

It uses custom data structures instead of built-in STL maps or priority queues.

Compilation & Execution

1. Compile all .cpp files together:

g++ main.cpp TreeNode.cpp FakeMap.cpp File.cpp Heap1.cpp Heap2.cpp HashMap.cpp -o program

3. Run the program:

./program

4. Enter commands interactively.

Data Structures

1. **TreeNode**

- Represents a version node of a file.

- Stores:

  - `version_id`

  - `content` (file content at this version)

  - `message` (snapshot message if any)

  - `created_timestamp`, `snapshot_timestamp`

  - `parent` (previous version)

  - `children` (branching versions)

  - `snapshot` flag (true if it’s a saved snapshot)

---

2. **fakemap**

- Lightweight array-based version mapping.

- Maps `version_id → TreeNode*`.

- Dynamically resizes vector to store pointers.

---

3. **File**

- Represents a file in the system.

- Stores:

  - Root node (initial version)

  - Active version pointer

  - `fakemap` for fast version lookup

  - Heap indices (`ind_in_heap1`, `ind_in_heap2`)

  - Metadata: total versions, last updated timestamp

- Supports operations:

  - `read()`

  - `insert(addit)`

  - `update(addit)`

  - `snapshot(message)`

  - `rollback([version_id])`

  - `history()`

---

4. **heap1**

- Max-heap based on **total versions**.

- Used for answering **MOST_VERSIONS** query.

- Maintains index of each file inside the heap for O(log n) updates.

---

5. **heap2**

- Max-heap based on last **updated timestamp**.

- Used for answering **RECENT_FILES** query.

- Maintains index for O(log n) reordering.

---

6. **HashMap**

- Custom hash table mapping `filename → File*`.

- Uses:

  - Polynomial rolling hash with base `P=31` and modulo `1e9+9`.

  - Separate chaining with vectors for collisions.

- Provides:

  - `put(key, value)`

  - `get(key)`

  - `contains(key)`

Commands Supported

File Operations

- CREATE → Creates a new file.

- READ → Displays current content of the file.

- INSERT "text" → Appends text to current version.

- UPDATE "text" → Replaces content with text.

- SNAPSHOT "message" → Marks current version as snapshot with message.

- ROLLBACK → Goes back to parent version.

- ROLLBACK → Jumps to specific version.

- HISTORY → Shows version history with timestamps.

Queries Across Files

- RECENT_FILES → Prints the `k` most recently updated files.

- MOST_VERSIONS → Prints the `k` files with most versions.

System

- EXIT → Exits the program.

Design Decisions

- Tree structure for versioning: allows branching rollbacks.

- **NOTE THAT THE INPUT MESSAGE/TEXT IS TO BE PROVIDED IN "" WHEREVER REQUIRED.**

- Custom fakemap: faster version lookup than searching history.

- Heaps with index tracking: efficient top-k queries while maintaining update consistency.

- Custom HashMap: avoids STL `unordered_map`, demonstrates hashing implementation.

- Snapshots: ensure version immutability (once a snapshot is saved, new edits create a new version).