Design Decisions in MiniGit

1. Philosophy Behind Design Decisions

MiniGit was intentionally designed as a **learning-oriented version control system**, not as a full replacement for Git. Every design decision was guided by three core principles:

1. **Simplicity over completeness** – The goal was clarity, not feature overload.
2. **Transparency over abstraction** – Data is stored in readable formats to make internals visible.
3. **Conceptual alignment with Git** – Even simplified features mirror real Git ideas.

These principles ensured that MiniGit acts as both a functional tool and an educational system.

---

2. Why File-Based Storage Was Chosen

Decision

Use the **local file system** as the primary persistence layer instead of databases or in-memory-only structures.

Reasoning

- Version control systems fundamentally manage files.
- File-based storage closely resembles Git's `.git` directory.
- It allows inspection using simple OS commands.

Benefits

- Easy debugging and verification
- Persistence across program executions
- No dependency on external libraries

Trade-Offs

- Slower than in-memory storage
- Requires careful file handling

Despite trade-offs, this choice reinforces conceptual clarity.

---

3. Why Text Files Instead of Binary Formats

Decision

Store metadata (commits, branches, index) as **plain text files**.

Reasoning

- Human-readable format aids learning
- Easy manual inspection
- No need for serialization libraries

Example

A commit file contains:

```
parent: <hash>
branch: main
message: Initial commit
file1.txt: <hash>
```

Comparison to Git

Git	MiniGit
Binary compressed objects	Plain text objects
High efficiency	High readability

---

4. Hash-Based Identification of Objects

Decision

Use hashing to uniquely identify commits and file contents.

Reasoning

- Ensures uniqueness
- Avoids name conflicts
- Models Git’s content-addressable storage

Why std::hash

- Built-in C++ support
- Fast execution

- Adequate for learning-level collision risk

Trade-Off

- Not cryptographically secure (unlike SHA-1/SHA-256 in Git)

This trade-off is acceptable since MiniGit is educational, not security-critical.

5. Commit Structure as a Linked List

Decision

Each commit stores a reference to its **parent commit hash**.

Why This Model

- Enables linear history traversal
- Mimics Git commit DAG in simplified form
- Simplifies log traversal

Benefits

- Easy implementation
- Clear mental model
- Efficient backward traversal

Limitations

- No merge commits yet (single parent only)

This design lays the foundation for future DAG expansion.

6. Why unordered_map Was Preferred

Decision

Use `unordered_map` for staging area and metadata mapping.

Reasoning

- O(1) average lookup time
- Natural key-value mapping (filename → hash)
- Order is irrelevant

Comparison

Data Structure	Reason Rejected
vector	Slow lookup
map	O(log n) overhead

This choice optimizes performance while keeping code readable.

---

7. Branch Representation Design

Decision

Branches stored as:

```
branch_name: commit_hash
```

Why This Works

- Simple pointer-based abstraction
- Mirrors Git's `refs/heads`
- Allows instant branch switching

HEAD Management

- `HEAD.txt` stores active branch name
- Checkout updates HEAD only

This separation simplifies branch logic.

---

8. CLI-First Design Choice

Decision

Expose all functionality via a **command-line interface**.

Reasoning

- Matches Git's usage model
- Encourages scripting and automation
- Minimal UI overhead

Commands Implemented

- init
- add
- commit
- log
- branch
- checkout

This approach ensures extensibility.

---

9. Error Handling Strategy

Decision

Fail gracefully with informative messages.

Examples

- Invalid command → "Unknown command"
- Empty history → "No commits yet"

Why This Matters

- Improves user experience
- Aids debugging
- Prevents silent failures

---

10. Scalability and Future Design Choices

Although MiniGit is simple, it was designed with extensibility in mind:

- Commit DAG support
- Merge commits
- Diff and status commands
- GUI or Web interface

Current design decisions ensure minimal refactoring for future growth.

---

11. Summary of Key Design Decisions

Decision	Justification
File-based storage	Persistence + Git alignment
Text metadata	Readability
Hash identifiers	Uniqueness
Linked-list commits	Simple history
unordered_map	Performance
CLI interface	Authentic VCS usage

These decisions collectively form a clean, educational, and professional system.