## ****Initialization — Presentation Guide****

### 🎯 ****Slide 1: Project Overview****

## ****What is MiniGit?****

**MiniGit** is a lightweight version control system built in **C++**, inspired by **Git**. It replicates core Git functionalities such as:

* init – Initialize a new repository
* add – Stage file(s)
* commit – Save a snapshot of the project
* log – View commit history
* branch – Create branches
* checkout – Switch to a different branch or commit
* merge – Combine changes from different branches
* diff – Compare file differences

It uses **SHA-1 hashing**, **file system operations**, and various **data structures** to manage the repository in a folder named .minigit/.

## 🔧 ****Tools & Technologies Used****

| **Tool/Library** | **Purpose** |
| --- | --- |
| C++ | Core programming language |
| <filesystem> | For directory and file creation |
| <fstream> | For reading/writing files |
| <unordered\_map>, <unordered\_set> | Storing and accessing data efficiently |
| OpenSSL (SHA-1) | Hashing file contents for version tracking |

* .

### 📂 ****Slide 3: Folder Structure Created****

.minigit/

│

├── HEAD → points to current branch (e.g., refs/master)

├── index → tracks staged files (like Git index)

├── objects/ → stores blobs, trees, and commit objects

└── refs/

└── master → points to latest commit in 'master' branch

## ****Key Internal Concepts****

### 📌 1. ****Blob****

* A **blob** is a saved version of a file.
* Stores the **contents only**, identified using **SHA-1 hash**.
* Saved under .minigit/objects/.

### 📌 2. ****Tree****

* A tree represents a **directory** and contains a list of blobs or other trees (subfolders).
* Used to track the folder structure and file hierarchy.

### 📌 3. ****Commit****

* A snapshot of the project at a moment in time.
* Contains:
  + A reference to the **tree**
  + A **parent commit** (or two in case of merge)
  + A **timestamp**
  + A **commit message**

### 📌 4. ****HEAD****

* Points to the current branch (e.g., refs/master), which in turn points to the latest commit hash.

### 📌 5. ****Refs****

* Keeps track of branches by storing the hash of the latest commit.

## 🧠 ****DSA (Data Structures and Algorithms) Used****

| **DSA Used** | **Purpose** |
| --- | --- |
| unordered\_map | - Used in staging\_area to map file paths to blob hashes. - Used to collect file lists for tree and commit traversal. |
| unordered\_set | - Used in find\_lca() to track commit ancestors efficiently. |
| vector | - Used to hold entries of a directory (trees/blobs). - Helps sort and serialize directory content. |
| **DAG (Directed Acyclic Graph)** | - The commit history is a DAG — each commit points to its parent(s). - Used in merge and log traversal. |
| **Hashing (SHA-1)** | - Generates unique IDs for files, trees, and commits. - Avoids redundant storage. |
| **Recursion** | - Used in collect\_files() to recursively traverse the tree structure. |
| **File I/O** | - Used in read\_file() and write\_file() to persist objects in .minigit/objects/. |

## ⚙️ ****Implementation Flow (Features)****

### init()

* Creates .minigit/, HEAD, index, objects/, and refs/.
* Sets up the version control structure.

### add(path)

* Reads file content, hashes it (SHA-1), and stores it as a blob.
* Updates the index and builds the staged\_trees.

### commit(message)

* Builds tree objects from the staging area.
* Saves commit metadata (tree hash, parent commit, message, time).
* Clears staging area after committing.

### log()

* Reads HEAD and prints the commit history by tracing parent commits.

### branch(name)

* Saves a new reference in .minigit/refs/<name> pointing to the current commit.

### ✅ checkout(branch|commit)

* Loads the state of a commit or branch into the working directory.
* Rebuilds the staging area to match the checked-out state.

### merge(branch)

* Finds the **lowest common ancestor (LCA)** of two commits.
* Merges changes from both commits, detects conflicts.
* Creates a new commit with two parents if successful.

### diff(commit1, [commit2])

* Compares file content differences between two commits or between a commit and the working directory.

## 💡 ****Why This Project Matters****

* Helps you understand how real-world tools like **Git** work internally.
* Sharpens your skills in:
  + File handling and directories
  + Hashing and SHA-1
  + Tree structures and DAGs
  + Designing and simulating real software systems
* Prepares you for systems-level programming and interviews.

### 💻 ****Slide 4: Code Behind**** init()

### ⚙️ ****Slide 2: What Happens During**** init

"The init command sets up the internal structure of the MiniGit repository, just like git init does."

#### ✅ ****Objective of**** init:

* Create a .minigit/ folder structure to store version control data
* Set the default branch to master

Prepare the workspace for future operations like add, commit, etc

Here’s the exact code for init() from your class:

public:

    void init() {

        fs::create\_directories(objects\_dir); //// Creates .minigit/objects

        fs::create\_directories(refs\_dir); // Creates .minigit/refs

        write\_file(head\_file, "ref: refs/master");// Sets HEAD to point to master branch

        write\_file(index\_file, "");// Initializes empty index

        std::cout << "Initialized MiniGit repo\n";

    }

* init() creates the MiniGit equivalent of Git’s internal structure.
* Without this, the system has no place to store commits, trees, or track branches.
* Think of .minigit/ as the brain of the entire version control system.

### 🧠 ****Slide 5: What Each Line Does****

| **Code Line** | **Explanation** |
| --- | --- |
| fs::create\_directories(objects\_dir); | Creates the directory .minigit/objects/, where all file contents (blobs), directory trees, and commits will be saved using SHA-1 hashes. |
| fs::create\_directories(refs\_dir); | Creates .minigit/refs/, where each branch (like master) will be represented as a file containing the latest commit hash. |
| write\_file(head\_file, "ref: refs/master"); | Creates the .minigit/HEAD file and sets it to reference refs/master, indicating the default branch. |
| write\_file(index\_file, ""); | Creates an empty index file to keep track of files that have been added but not yet committed (i.e., staging area). |
| std::cout << "Initialized MiniGit repo\n"; | Prints confirmation to the user. |

### 🧱 ****Slide 6: Why Initialization Matters****

* **Foundation of the Repository**:
  + All future operations (add, commit, branch, etc.) depend on these folders and files.
  + HEAD determines the current working branch.
  + refs/ keeps track of each branch’s latest commit.
  + objects/ stores every piece of content permanently (blobs, trees, commits).
  + index stages content before committing.