
# ADDIS ABABA INSTITUTE OF TECHNOLOGY

## Software engineering department

Fundamental of Data Structure and Algorithm Analaysis

(SECT-3091)

# Project Title: MiniGit: A Custom Version Control System Codebase with modular functions and comments


Name Id

1. Getamesay Hailemichael……....ATE/5152/13
2. Sisay Leykun.....ATE/0493/15
3. Tamrat Arage.....ATE/8888/15


Submitted to**: Dr. Beakal Gizachew Assefa**

Submissin date: June 24, 2025

# Blob Module — Handles file content storage

## Blob.h

```
#pragma once

#include <string>

// Blob handles file content storage by hash
class Blob {
public:
    // Compute hash of file content
    static std::string computeHash(const std::string& content);

    // Store file content as blob
    static void storeBlob(const std::string& content, const std::string& hash);

    // Read stored blob content by hash
    static std::string readBlob(const std::string& hash);
};
```

## Blob.cpp

```
#include "Blob.h"
#include <fstream>
#include <filesystem>
#include <functional>

// Hash file content using standard hash function
std::string Blob::computeHash(const std::string& content) {
    std::hash<std::string> hasher;
    return std::to_string(hasher(content));
}

// Store file content into blob object storage
void Blob::storeBlob(const std::string& content, const std::string& hash) {
    std::filesystem::create_directories(".minigit/objects");
    std::ofstream out(".minigit/objects/" + hash);
    out << content;
    out.close();
}

// Retrieve stored blob content by hash
std::string Blob::readBlob(const std::string& hash) {
    std::ifstream in(".minigit/objects/" + hash);
    if (!in) return "";
    return std::string((std::istreambuf_iterator<char>(in)), std::istreambuf_iterator<char>());
}
```

# Commit Module — Commit history management

## Commit.h

```
#pragma once

#include <string>
#include <map>

// Commit represents a snapshot of repository state
class Commit {
public:
   // Create a commit with metadata and file map
   static std::string createCommit(const std::string& parent,
                                   const std::map<std::string, std::string>& files,
                                   const std::string& message);

   // Debug utility: read raw commit file
   static void readCommit(const std::string& hash);
};
```

## Commit.cpp

```
#include "Commit.h"
#include <iostream>
#include <fstream>
#include <filesystem>
#include <ctime>

// Create commit file containing metadata + file map
std::string Commit::createCommit(const std::string& parent, const std::map<std::string, std::string>& files, const
std::string& message) {
    std::string commitData = parent + "\n" + message + "\n" + std::to_string(std::time(0)) + "\n";
    for (const auto& file : files) {
        commitData += file.first + " " + file.second + "\n";
    }

    std::hash<std::string> hasher;
    std::string commitHash = std::to_string(hasher(commitData));

    std::filesystem::create_directories(".minigit/commits");
    std::ofstream out(".minigit/commits/" + commitHash);
    out << commitData;
    out.close();

    return commitHash;
}

// Read raw commit content (for debugging)
void Commit::readCommit(const std::string& hash) {
    std::ifstream in(".minigit/commits/" + hash);
    if (!in) {
        std::cout << "Commit not found." << std::endl;
        return;
    }
    std::string line;
    while (getline(in, line)) {
        std::cout << line << std::endl;
    }
}
```

# Branch Module — Branch reference handling

## Branch.h

```
#pragma once

#include <string>

// Branch manages branch reference files
class Branch {
public:
    // Create or update branch pointer
    static void createBranch(const std::string& name, const std::string& commitHash);

    // Get current commit for given branch
    static std::string getBranchCommit(const std::string& name);
};
```

## Branch.cpp

```
cpp
CopyEdit
#include "Branch.h"
#include <fstream>
#include <filesystem>

// Create branch by writing reference file
void Branch::createBranch(const std::string& name, const std::string& commitHash) {
    std::filesystem::create_directories(".minigit/refs");
    std::ofstream out(".minigit/refs/" + name);
    out << commitHash;
    out.close();
}

// Read branch reference file
std::string Branch::getBranchCommit(const std::string& name) {
    std::ifstream in(".minigit/refs/" + name);
    if (!in) return "";
    std::string commitHash;
    getline(in, commitHash);
    return commitHash;
}
```

# Staging Area — Temporary file staging

## StagingArea.h

```
#pragma once

#include <string>
#include <set>

// Staging area holds files added for next commit
class StagingArea {
private:
    std::set<std::string> files;

public:
    void add(const std::string& filename);
    void clear();
    const std::set<std::string>& getFiles() const;
};
```

## StagingArea.cpp

```
#include "StagingArea.h"

void StagingArea::add(const std::string& filename) {
    files.insert(filename);
}

void StagingArea::clear() {
    files.clear();
}

const std::set<std::string>& StagingArea::getFiles() const {
    return files;
}
```

# Core MiniGit System

## MiniGitSystem.h

```
#pragma once

#include "StagingArea.h"
#include <string>
#include <map>

// The core MiniGit version control system
class MiniGitSystem {
private:
    std::string currentBranch;
    StagingArea stagingArea;

    // Internal helpers
    void loadHead();
    void updateHead(const std::string& commitHash);
    std::map<std::string, std::string> loadCommitFiles(const std::string& commitHash);
    void restoreFilesFromCommit(const std::string& commitHash);
    void printCommitLog(const std::string& commitHash);
    void compareFileContents(const std::string& file, const std::string& content1, const std::string& content2);

public:
    MiniGitSystem();
    void init();
    void addFile(const std::string& filename);
    void commit(const std::string& message);
    void log();
    void branch(const std::string& branchName);
    void checkout(const std::string& name);
    void merge(const std::string& branchName);
    void diff(const std::string& commitHash1, const std::string& commitHash2);
};
```

## MiniGitSystem.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include "MiniGitSystem.h"
#include "Blob.h"
#include "Commit.h"
#include "Branch.h"

#include <iostream>
#include <fstream>
#include <filesystem>
#include <sstream>
#include <ctime>

MiniGitSystem::MiniGitSystem() { loadHead(); }

// Load HEAD branch info
void MiniGitSystem::loadHead() {
    std::ifstream in(".minigit/HEAD");
    if (in) getline(in, currentBranch);
    else currentBranch = "";
}

// Update HEAD pointer after commit
void MiniGitSystem::updateHead(const std::string& commitHash) {
    if (currentBranch.empty()) {
        std::ofstream(".minigit/HEAD") << "master";
        currentBranch = "master";
    }
    Branch::createBranch(currentBranch, commitHash);
}

// Initialize repository structure
void MiniGitSystem::init() {
    if (std::filesystem::exists(".minigit")) {
        std::cout << "Repository already initialized." << std::endl;
        return;
    }
    std::filesystem::create_directories(".minigit/objects");
    std::filesystem::create_directories(".minigit/commits");
    std::filesystem::create_directories(".minigit/refs");
    std::ofstream(".minigit/HEAD") << "master";
    std::ofstream(".minigit/refs/master");
    std::cout << "Initialized empty MiniGit repository." << std::endl;
    loadHead();
}

// Stage file for commit
void MiniGitSystem::addFile(const std::string& filename) {
    if (!std::filesystem::exists(filename)) {
        std::cout << "File not found." << std::endl;
        return;
    }
```

```
    stagingArea.add(filename);
    std::cout << "File staged: " << filename << std::endl;
}

// Commit staged files
void MiniGitSystem::commit(const std::string& message) {
    std::map<std::string, std::string> filesMap;
    std::string parentCommit = Branch::getBranchCommit(currentBranch);
    if (!parentCommit.empty()) filesMap = loadCommitFiles(parentCommit);

    // Process staged files
    for (const auto& file : stagingArea.getFiles()) {
        std::ifstream inFile(file);
        std::string content((std::istreambuf_iterator<char>(inFile)), std::istreambuf_iterator<char>());
        std::string hash = Blob::computeHash(content);
        Blob::storeBlob(content, hash);
        filesMap[file] = hash;
    }

    std::string commitHash = Commit::createCommit(parentCommit, filesMap, message);
    updateHead(commitHash);
    stagingArea.clear();
    std::cout << "Commit created: " << commitHash << std::endl;
}

// Load commit file mappings
std::map<std::string, std::string> MiniGitSystem::loadCommitFiles(const std::string& commitHash) {
    std::ifstream in(".minigit/commits/" + commitHash);
    std::string parent, msg, timestamp, line;
    getline(in, parent); getline(in, msg); getline(in, timestamp);
    std::map<std::string, std::string> filesMap;
    while (getline(in, line)) {
        size_t pos = line.find(" ");
        if (pos != std::string::npos)
            filesMap[line.substr(0, pos)] = line.substr(pos + 1);
    }
    return filesMap;
}

// Display commit logs
void MiniGitSystem::log() {
    std::string commitHash = Branch::getBranchCommit(currentBranch);
    while (!commitHash.empty()) {
        printCommitLog(commitHash);
        commitHash = loadCommitFiles(commitHash)[""];
    }
}

// Print individual commit log entry
void MiniGitSystem::printCommitLog(const std::string& commitHash) {
    std::ifstream in(".minigit/commits/" + commitHash);
    std::string parent, message, timestamp;
    getline(in, parent); getline(in, message); getline(in, timestamp);
    std::time_t ts = std::stoll(timestamp);
```

```
    char buf[26];
    ctime_s(buf, sizeof(buf), &ts);
    std::cout << "Commit: " << commitHash << "\nDate: " << buf << "Message: " << message << "\n----------
\n";
}

// Create new branch
void MiniGitSystem::branch(const std::string& branchName) {
    std::string currentCommit = Branch::getBranchCommit(currentBranch);
    Branch::createBranch(branchName, currentCommit);
    std::cout << "Branch created: " << branchName << std::endl;
}

// Checkout branch or commit
void MiniGitSystem::checkout(const std::string& name) {
    std::string targetCommit = Branch::getBranchCommit(name);
    if (targetCommit.empty() && std::filesystem::exists(".minigit/commits/" + name))
        targetCommit = name;
    if (targetCommit.empty()) {
        std::cout << "Branch or commit not found." << std::endl;
        return;
    }
    restoreFilesFromCommit(targetCommit);
    if (std::filesystem::exists(".minigit/refs/" + name)) {
        currentBranch = name;
        std::ofstream(".minigit/HEAD") << name;
    }
    std::cout << "Checked out to: " << name << std::endl;
}

// Restore working directory from commit snapshot
void MiniGitSystem::restoreFilesFromCommit(const std::string& commitHash) {
    auto filesMap = loadCommitFiles(commitHash);
    for (const auto& [file, hash] : filesMap) {
        std::ofstream out(file);
        out << Blob::readBlob(hash);
    }
}

// Merge feature (not implemented)
void MiniGitSystem::merge(const std::string& branchName) {
    std::cout << "Merge feature not implemented yet." << std::endl;
}

// Diff between two commits
void MiniGitSystem::diff(const std::string& commitHash1, const std::string& commitHash2) {
    auto files1 = loadCommitFiles(commitHash1);
    auto files2 = loadCommitFiles(commitHash2);
    std::set<std::string> allFiles;
    for (auto& [f, _] : files1) allFiles.insert(f);
    for (auto& [f, _] : files2) allFiles.insert(f);

    for (auto& file : allFiles) {
        std::string content1 = files1.count(file) ? Blob::readBlob(files1[file]) : "";
```

```
        std::string content2 = files2.count(file) ? Blob::readBlob(files2[file]) : "";
        if (content1 == content2) {
            std::cout << "File: " << file << " — No changes." << std::endl;
        } else {
            std::cout << "File: " << file << " — Differences:" << std::endl;
            compareFileContents(file, content1, content2);
        }
    }
}

// Compare file contents line-by-line
void MiniGitSystem::compareFileContents(const std::string& file, const std::string& content1, const std::string&
content2) {
    std::istringstream s1(content1), s2(content2);
    std::string l1, l2;
    int line = 1;
    while (true) {
        bool b1 = (bool)getline(s1, l1);
        bool b2 = (bool)getline(s2, l2);
        if (!b1 && !b2) break;
        if (l1 != l2)
            std::cout << "Line " << line << ":\n  Commit1: " << l1 << "\n  Commit2: " << l2 << std::endl;
        line++;
    }
}
```

# main.cpp

```
#include "MiniGitSystem.h"
#include <iostream>

int main() {
    MiniGitSystem system;
    std::string command;
    std::cout << "Welcome to MiniGit!\n";
    while (true) {
        std::cout << "\nMiniGit > ";
        std::cin >> command;
        if (command == "init") system.init();
        else if (command == "add") {
            std::string filename; std::cin >> filename;
            system.addFile(filename);
        }
        else if (command == "commit") {
            std::cin.ignore();
            std::string msg;
            std::cout << "Enter commit message: ";
            std::getline(std::cin, msg);
            system.commit(msg);
        }
        else if (command == "log") system.log();
        else if (command == "branch") {
            std::string name; std::cin >> name;
            system.branch(name);
        }
        else if (command == "checkout") {
            std::string name; std::cin >> name;
            system.checkout(name);
        }
        else if (command == "diff") {
            std::string c1, c2; std::cin >> c1 >> c2;
            system.diff(c1, c2);
        }
        else if (command == "exit") break;
        else std::cout << "Unknown command.\n";
    }
    return 0;
}
```