**B202 Advance Programming**

**Project Report**

**Project: Version Control System**

**Word Count:**

Table of Contents

[Introduction: 2](#_Toc153415687)

[Features: 2](#_Toc153415688)

[Key Concepts: 2](#_Toc153415689)

[Repository: 2](#_Toc153415690)

[Staging Area: 2](#_Toc153415691)

[Commit: 2](#_Toc153415692)

[Revert: 2](#_Toc153415693)

[Code Structure: 2](#_Toc153415694)

[Header Files: 2](#_Toc153415695)

[Color Definitions: 2](#_Toc153415696)

[Main Function: 2](#_Toc153415697)

[Git Command Handling: 3](#_Toc153415698)

[Git Class Object: 3](#_Toc153415699)

[Command Execution: 3](#_Toc153415700)

[Console Output: 4](#_Toc153415701)

[Code Examples: 4](#_Toc153415702)

[Repository: 4](#_Toc153415703)

[Staging Area: 5](#_Toc153415704)

[Commit: 6](#_Toc153415705)

[Revert: 7](#_Toc153415706)

[GitLog: 7](#_Toc153415707)

[Conclusion: 8](#_Toc153415708)

# Introduction:

Version control system, also known as source control, it is used to track and manage changes in source code. It is the tool which helps to manage the modification/changes to source code over time. As development environments have accelerated, version control system helps to work fast and smart, it also reduces development time and rise successful deployments.

# Features:

* This project is a simple clone of Git.
* Add code to staging area.
* Make commits to create different versions of source code file of the project.
* Display all the commits.
* It can revert to a last version of commit.
* Easy to track changes.
* Easy to track the history of project files.

# Key Concepts:

## Repository:

It stores the complete track of the source code and project files and allows user to track changes.

## Staging Area:

In this area user is allowed specifically choose the changes which is required to include in next commit.

## Commit:

It shows changes made in the files. Each commit contains unique identifier and user information, timestamp, and a commit message.

## Revert:

User can fix mistakes and bugs or revert unwanted changes without losing the history of commits.

# Code Structure:

## Header Files:

"Class.cpp" and other required header files are included in the code. The implementation of the gitClass class, which encapsulates the Git functionality, is probably found in the "Class.cpp" file.

## Color Definitions:

The code uses escape sequences to specify several color constants. These constants are used to enhance the console output's visual appeal by adding colour.

## Main Function:

The program's entry point serves as its main purpose. To find the appropriate Git command and its parameters, it requires two command-line arguments: argc and argv.

## Git Command Handling:

To handle various Git instructions, the code employs conditional statements (if and else if). To ascertain the intended action, it verifies the value of the first command-line input (argv[1]).

## Git Class Object:

At the start of the main function, an instance of the gitClass class is created. Invoking the Git functionality with this object depends on the command-line arguments that are supplied.

## Command Execution:

The code executes the specified action by calling the relevant member functions of the gitClass object in accordance with the given command. To initialize a new Git repository, for instance, the gitInit function is executed if the command is "init."

## Console Output:

The code displays instructions and messages to the user using cout commands. To provide color to the output, it additionally makes use of the previously defined color constants.

# Code Examples:

## Repository:

A screen shot of a computer code

Description automatically generated

Figure 1: Initializing a Git Repository

The gitClassObj object is constructed and the command-line option is verified in this section of code. The gitInit method is executed to start a new Git repository if the parameter is "init". After that, a success message appears.

## Staging Area:

A computer screen shot of text

Description automatically generated

Figure 2: Adding Files to the Staging Area

The "add" command is handled by this bit of code. It determines whether the right number of arguments is supplied. If not, an error message specifying the proper usage is displayed. The code determines whether a parameter is "." (which indicates all files) or a specific file name if the arguments are valid. Next, the function gitAdd is invoked with the relevant arguments.

## Commit:

A computer screen with text

Description automatically generated

Figure 3: Committing changes

The "commit" instruction is handled by this bit of code. It determines whether the right number of arguments is supplied. If not, an error message specifying the proper usage is displayed. The function determines whether the second parameter, which denotes a commit message, is "-m" if the arguments are valid. The commit message is then passed to the gitCommit function, which returns a success message.

## Revert:

A computer screen shot of text

Description automatically generated

Figure 4: Reverting to a Previous Commit

The "revert" command is handled by this bit of code. It determines whether the right number of arguments is supplied. If not, an error message specifying the proper usage is displayed. The function verifies whether the second parameter, which indicates rolling back to the most recent commit, is "HEAD" or a particular commit hash if the arguments are valid. The relevant parameter is subsequently passed to the gitRevert function, and a success message is shown.

## GitLog:

A computer screen shot of a program code

Description automatically generated

Figure 5:Git Log

The command line argument supplied to the application is represented by the argument variable in the code snippet. The code runs the appropriate function after determining whether the parameter matches one or more specific Git commands.

# Conclusion:

In this project we looked at a C++ application that mimics the Git CLI in a more basic manner. We looked at the code structure and examples, and we talked about some of the important Git ideas. The provided code shows how to add files to the staging area, commit changes, and roll back commits. It also shows how to start a Git repository. Collaboration and version control in software development projects require an understanding of the Git CLI and its fundamental ideas.