Chapter 1

INTRODUCTION

1.1 INTRODUCTION:

A version control system (VCS) is a tool used by developers to manage changes to source code over time. It keeps track of modifications, facilitates collaboration among team members, and allows for reverting to previous versions if needed. Git is one of the most popular VCS, widely used for its flexibility, speed, and distributed nature, enabling multiple developers to work on the same project concurrently.

In the ever-evolving landscape of software development, Version Control Systems (VCS) play a pivotal role in managing the complexities of collaborative coding. This introduction unveils a mini project developed in C++, offering a robust VCS solution with both centralized and distributed functionalities.

1. Understanding Version Control Systems (VCS):

Version Control Systems are essential tools that track changes to source code over time, facilitating collaboration among developers, enabling code rollback, and ensuring project integrity. They come in two primary architectures: centralized and distributed.

2. Distributed and Centralized Architecture: The mini-project incorporates elements of both distributed and centralized VCS architectures, offering flexibility in deployment and workflow management.

- Distributed VCS: In a distributed setup, each user maintains a complete copy of the repository, including its entire history. This approach enhances redundancy, fault tolerance, and autonomy, as users can work offline and synchronize changes with remote repositories as needed.
- Centralized VCS: Alternatively, the system can operate in a centralized mode where a single repository serves as the authoritative source of truth. Users interact with this central repository to access project files and manage revisions. While centralized VCS streamlines coordination and access control, it may introduce single points of failure and dependencies on network connectivity.

1.2 LITERATURE REVIEW:

A literature review of version control systems (VCS) typically covers various aspects including the history, evolution, and different types of VCS, as well as their benefits, challenges, and applications in software development. It may delve into the comparison of different VCS platforms such as Git, Subversion, Mercurial, etc., highlighting their features, strengths, and weaknesses. Additionally, the literature review may explore topics like branching and merging strategies, best practices for using VCS in collaborative environments, and the impact of VCS on software quality, productivity, and project management. Research might also focus on emerging trends in VCS, such as the integration of VCS with continuous integration/delivery systems or the adoption of VCS in non-software domains. Overall, a literature review provides a comprehensive overview of the existing knowledge, research, and practices related to version control systems.

1.3 NEED OF PRESENT WORK

The need for the present work on version control systems (VCS) can be multifaceted and context-dependent. Here are some common reasons why research or development efforts in this area might be necessary:

1. **Advancing Technology:**
   With the continuous evolution of software development practices and technologies, there is a constant need to improve version control systems to meet the changing requirements of developers and organizations.
2. **Enhancing Collaboration:**
   Collaboration among team members is essential in modern software development. The present work may aim to enhance VCS capabilities to facilitate smoother collaboration, particularly in distributed teams or open-source projects.
3. **Improving Efficiency:**
   Efficient version control processes can significantly impact the productivity of development teams. The current work might focus on streamlining workflows, optimizing performance, or reducing overhead in VCS operations.
4. **Addressing Security Concerns:**
   Security is a critical aspect of software development, and VCS systems are not exempt from vulnerabilities. The present work may aim to address security risks associated with VCS usage, such as unauthorized access, data breaches, or malicious code injections.
5. **Supporting Diverse Workflows:**
   Different projects and organizations have unique workflows and requirements. The need for the present work might involve developing VCS features or tools that better support diverse development workflows, such as feature branching, release management, or continuous integration/continuous deployment (CI/CD).

Overall, the need for the present work on version control systems stems from the ongoing demand for better tools, practices, and technologies to support efficient, and secure software development processes.

1.4 OBJECTIVES OF THIS WORK:

1. **Tracking Changes:**
   One of the primary objectives is to track changes made to files and projects over time. This includes documenting who made the changes, when they were made, and what changes were implemented.

2. Versioning:
A VCS allows for creating different versions of a project or file. This enables developers to roll back to previous versions if necessary, compare different versions, and maintain a history of changes.

3.Backup and Recovery:
VCS serves as a backup mechanism by storing project history in a central repository or distributed across multiple locations. This ensures that even if files are lost or corrupted, they can be recovered from previous versions stored in the VCS.

4. Branching:
Enable the creation of branches, which are independent lines of development. Branching allows for experimentation and isolation of changes before they are merged back into the main codebase.

By fulfilling these objectives, version control systems play a crucial role in modern software development, manage project history, and ensure the quality and reliability of their codebases.

Chapter 2
PROBLEM STATEMENT

2.1 PROBLEM STATEMENT ON VERSION CONTROL SYSTEM

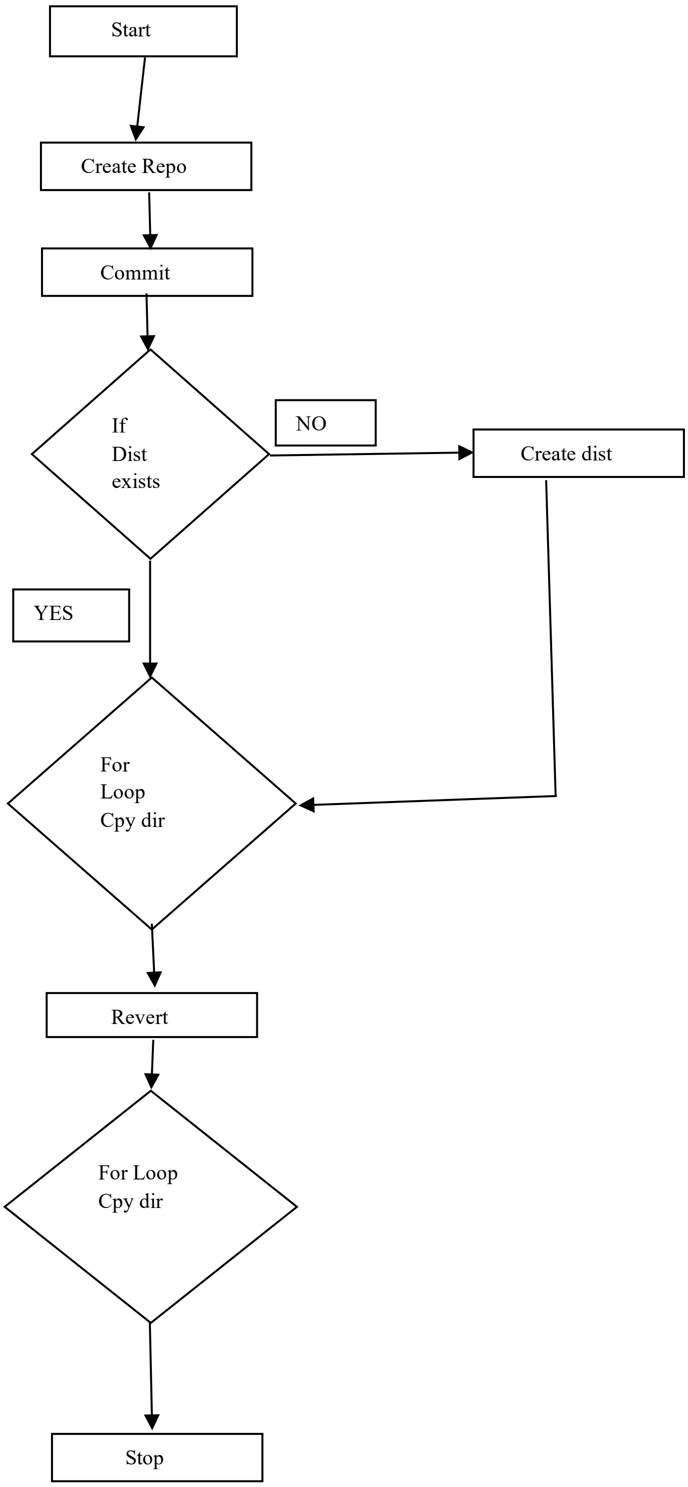
In the rapidly evolving landscape of software development, efficient management of source code and collaboration among developers is paramount. Traditional version control systems often face challenges related to scalability, fault tolerance, and distributed collaboration. To address these issues, the objective is to design and implement a Distributed Version Control System (DVCS) using C++.

The DVCS aims to overcome the limitations of centralized version control systems by decentralizing the repository and providing each user with a complete copy of the project history.

The DVCS will empower software development teams to collaborate effectively, manage code changes with confidence, and adapt to evolving project requirements with agility. The successful completion of this project will contribute to advancing the state-of-the-art in distributed version control systems and provide a valuable tool for developers seeking robust and scalable solutions for version control and collaboration.

Chapter 3
SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE



3.2 EXPLANATION

There are three main components :

1] Source Code.

2] Repository.

3] Command Line User Interface.

1. Source Code Directory:

Source code refers to the human-readable instructions written in a programming language that form the foundation of a software application. It's essentially the text-based representation of a program's logic, algorithms, and functionality. Programmers write and edit source code using text editors or integrated development environments (IDEs). Source code can be comprised of various programming languages such as Python, Java, C++, JavaScript, etc. It serves as the input to compilers or interpreters, which translate it into machine-readable code that computers can execute.

2. Repository:

A repository, often abbreviated as "repo," is a central storage location where version-controlled files and directories are stored, typically managed by a version control system (VCS). It serves as a single source of truth for a project, containing all its source code, documentation, configuration files, and other related assets. Repositories can be hosted locally on a developer's machine or remotely on servers such as GitHub, GitLab, Bitbucket, etc., allowing multiple developers to collaborate on the same codebase. A repository maintains a complete history of changes made to files over time, enabling developers to track revisions, revert to previous versions, and collaborate effectively.

3. Command Line User Interface(CLI):

A command-line user interface (CLI) is a text-based interface for interacting with a computer program or system through commands entered via a command-line interpreter. Instead of using graphical elements like windows, buttons, and menus, users interact with the program by typing commands and receiving text-based feedback. CLI tools are often preferred by developers and system administrators for their efficiency, flexibility, and automation capabilities. In the context of version control systems (VCS), CLI interfaces provide developers with a powerful way to interact with repositories, perform versioning operations (such as committing changes, branching, etc.), and manage their codebase from the command line.

Chapter 4
IMPLEMENTATION

4.1 IMPLEMENTATION STEPS

```
#include <iostream>
#include <filesystem>
#include <string>
#include <vector>
#include <unordered_map>

#include <thread>
#include <chrono>

#define RESET "\033[0m"
#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define BLUE "\033[34m"
#define MAGENTA "\033[35m"
#define CYAN "\033[36m"

using namespace std;
namespace fs = std::filesystem;

// Data structure to store commits for each branch
unordered_map<string, vector<string>> branch_commits;

void copy_directory(const fs::path &source, const fs::path &destination)
{
    if (!fs::exists(destination))
        fs::create_directories(destination);

    for (const auto &entry : fs::directory_iterator(source))
    {
        const auto &path = entry.path();
        const auto &new_path = destination / path.filename();

        cout << GREEN << "\nFiles ===>  ";
        cout << path << RESET;
        std::this_thread::sleep_for(std::chrono::milliseconds(10));

        if (fs::is_directory(path))
        {
            copy_directory(path, new_path);
```

```
        }
        else if (fs::is_regular_file(path))
            fs::copy_file(path, new_path, fs::copy_options::overwrite_existing);
    }
}

void display_branches()
{
    cout << CYAN << "\nAvailable branches:\n"
         << RESET;
    for (const auto &branch : branch_commits)
    {
        cout << branch.first << endl;
    }
}

void display_commits(const string &branch)
{
    cout << CYAN << "\nCommits in branch " << branch << ":\n"
         << RESET;
    for (const auto &commit : branch_commits[branch])
    {
        cout << commit << endl;
    }
}

void revert_commit(const string &branch, const string &commit)
{
    string source_dir = "dist/" + branch + "/" + commit;
    string dest_dir = "sourcecode";

    fs::path source_path(source_dir);
    fs::path dest_path(dest_dir);

    if (!fs::exists(source_path))
    {
        cout << RED << "\nCommit " << commit << " does not exist in branch " << branch <<
RESET << endl;
        return;
    }

    fs::remove_all(dest_path);              // Remove current source code
```

```
    copy_directory(source_path, dest_path); // Revert to commit

    cout << GREEN << "\n\nReverted to commit " << commit << " in branch " << branch <<
RESET << endl;
}

int main()
{
    string repo, branch, commit;
    int choice;
    while (true)
    {
        cout << RED <<
"\n\n========================================================\n\n"
             << RESET;
        cout << GREEN << "\t\tVersion Control System";
        cout << RED <<
"\n\n========================================================\n\n"
             << RESET;
        cout << CYAN << "\t\t1. Create Repository\n";
        cout << "\t\t2. Create Branch\n";
        cout << "\t\t3. Commit Changes\n";
        cout << "\t\t4. Revert Changes\n";
        cout << "\t\t5. Exit\n"
             << RESET;
        cout << RED << "\n\n________________________________________________________\n"
             << RESET;
        cout << CYAN << "Enter your choice: " << RESET;
        cin >> choice;

        if (choice == 1)
        {
            cout << CYAN << "\nEnter the repository name: " << RESET;
            cin >> repo;
            cout << GREEN << "Repository created: " << repo << RESET;
        }
        else if (choice == 2)
        {
            if (repo == "")
            {
                cout << RED << "\n Please create the repository first!\n"
                     << RESET;
```

```
            continue;
        }
        else
        {
            cout << CYAN << "\nEnter the branch name: " << RESET;
            cin >> branch;
            cout << GREEN << "Branch created: " << branch << RESET;
            // Initialize the branch with an empty vector of commits
            branch_commits[branch] = vector<string>();
        }
    }
    else if (choice == 3)
    {
        if (repo == "" && branch == "")
        {
            cout << RED << "\n Please create the repository and branch first!\n"
                 << RESET;
            continue;
        }
        else
        {
            cout << CYAN << "\nEnter the commit name: " << RESET;
            cin >> commit;

            string dest_dir = "dist/" + branch + "/" + commit;
            fs::path source_path("sourcecode");
            fs::path dest_path(dest_dir);

            copy_directory(source_path, dest_path);

            cout << GREEN << "\n\nAll changes are made!";
            cout << "\n\nCheck the following directory:\n"
                 << RESET;
            cout << MAGENTA << dest_dir << RESET;

            // Store the commit in the branch's commit list
            branch_commits[branch].push_back(commit);
        }
    }
    else if (choice == 4)
    {
        if (repo == "" || branch == "")
```

```
            {
                cout << RED << "\nPlease create the repository and branch first!\n"
                     << RESET;
                continue;
            }
            else
            {
                display_branches();
                cout << CYAN << "\nEnter the branch name: " << RESET;
                cin >> branch;

                display_commits(branch);
                cout << CYAN << "\nEnter the commit name to revert: " << RESET;
                cin >> commit;

                revert_commit(branch, commit);
            }
        }
        else if (choice == 5)
        {
            cout << CYAN << "\n\t Thank you for using the version control system!\n"
                 << RESET;
            cout << CYAN <<
"\n==============================================================\n"
                 << RESET;
            break;
        }
        else
        {
            cout << RED << "\nPlease enter a valid choice!\n"
                 << RESET;
            continue;
        }
    };

    return 0;
}
```

Chapter 5
REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

- Processor: Any modern processor capable of running the operating system is sufficient.
- Memory (RAM): A minimum of 500 KB RAM is recommended for smooth performance
- Storage: The system itself requires very little storage space. Even a few megabytes of disk space would be more than enough.

5.2 SOFTWARE REQUIREMENTS

Operating System: The system works on any modern operating system, including:

- Windows 7, 8, 10, or later versions.
- Linux distributions such as Ubuntu, Fedora, or CentOS
- Mac OS

C++ compiler

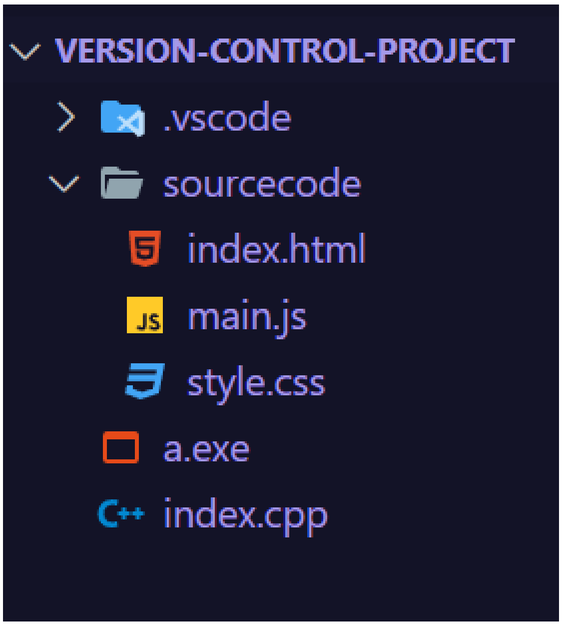
C++ v17 or greater

Chapter 6

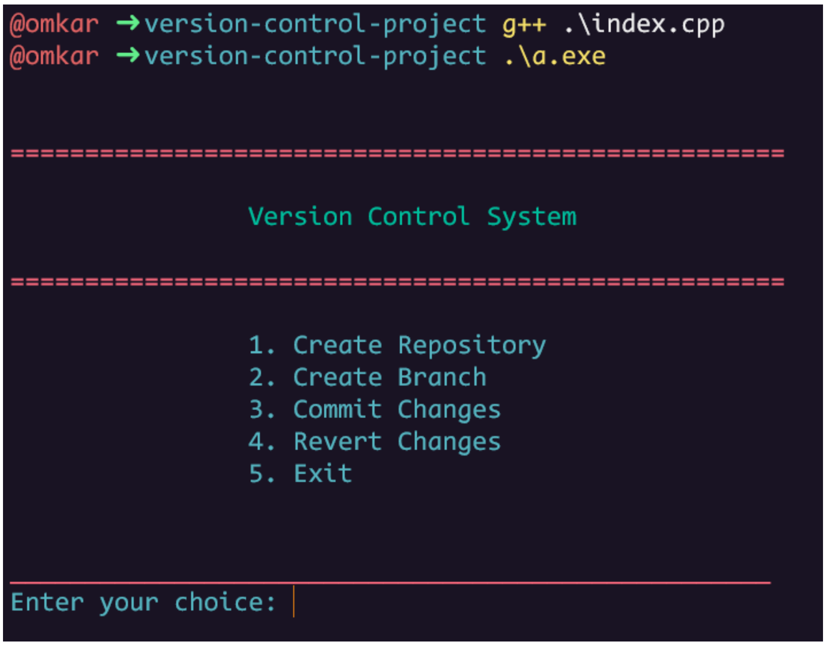
RESULT ANALYSIS AND FUTURE WORK

OUTPUT:-

Folder Structure Before Committing The Code



Code Execution:



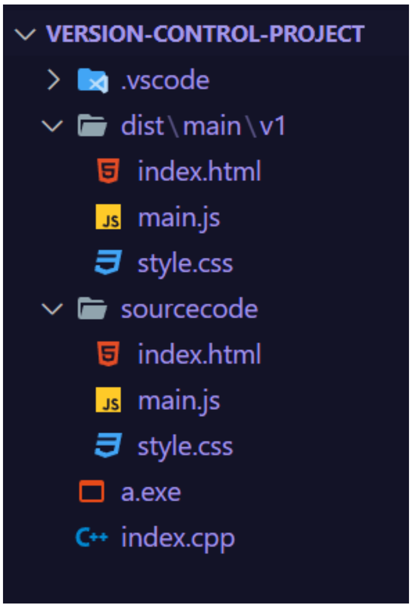
```
@omkar ➜version-control-project g++ .\index.cpp
@omkar ➜version-control-project .\a.exe


====================================================

                Version Control System

====================================================

                1. Create Repository
                2. Create Branch
                3. Commit Changes
                4. Revert Changes
                5. Exit


Enter your choice:
```

```
Enter your choice: 1

Enter the repository name: Portfolio-Website
Repository created: Portfolio-Website

====================================================

                Version Control System

====================================================

                1. Create Repository
                2. Create Branch
                3. Commit Changes
                4. Revert Changes
                5. Exit


Enter your choice:
```

```
Enter your choice: 2

Enter the branch name: main
Branch created: main

====================================================

                Version Control System

====================================================

                1. Create Repository
                2. Create Branch
                3. Commit Changes
                4. Revert Changes
                5. Exit


Enter your choice:
```

```
Enter your choice: 3

Enter the commit name: v1

Files ===>  "sourcecode\\index.html"
Files ===>  "sourcecode\\main.js"
Files ===>  "sourcecode\\style.css"

All changes are made!

Check the following directory:
dist/main/v1

====================================================

                Version Control System

====================================================

                1. Create Repository
                2. Create Branch
                3. Commit Changes
                4. Revert Changes
                5. Exit


Enter your choice:
```

Folder Structure after 1st commit as ‘v1’



Folder Structure after 2nd commit as 'v2'

```
VERSION-CONTROL-PROJECT
  > .vscode
  v dist\main
    > v1
    > v2
  v sourcecode
      index.html
      main.js
      style.css
    a.exe
    index.cpp
```

Reverting The Code:

```
====================================================

              Version Control System

====================================================

              1. Create Repository
              2. Create Branch
              3. Commit Changes
              4. Revert Changes
              5. Exit

___________________________________________________
Enter your choice: 4

Available branches:
main

Enter the branch name: main

Commits in branch main:
v1
v2

Enter the commit name to revert: v1

Files ===>  "dist/main/v1\\index.html"
Files ===>  "dist/main/v1\\main.js"
Files ===>  "dist/main/v1\\style.css"

Reverted to commit v1 in branch main
```

6.2 CONCLUSION

This version control system provides mechanisms for managing changes and ensuring the integrity of project versions.
By implementing robust methodologies and architecture, VCS enables teams to work efficiently, track progress, and maintain code quality throughout the development lifecycle.

6.3 FUTURE SCOPE

Future enhancements may include:

- Multi-developer Collaboration.
- Code Sharing.
- Merging multiple branches.
- Commit delete feature.
- Pushing the user interface from command line to graphical user interface.

REFERENCES

REFARANCE:

- 3rd International Conference on Computer Science and Computational Intelligence 2018 Research Paper

- Geeks for Geeks