VERSION CONTROL SYSTEM

Team Members:

Anurag Ghosh	- 2022202023
Aryan Gupta	- 2022202028
Prakhar Gupta	- 2022202027
Karan Bhatt	- 2022202003

Project Description:

Implemented a “git” like version control system with limited functionalities. The version control system is used to create and initialize a .git directory with necessary files and details, add files to the staging area and commit, maintain all versions of the project along with their commit-ids, check status and log, roll back to any previous version(commit).

Git link:

https://github.com/kbbhatt04/Version_Control_System

Requirements:

- g++ compiler (to compile the code)
- SHA1.hpp (Imported SHA1 library. File included in the folder)
- Linux

Architecture:

1. Init: (Prakhar Gupta)

The git init command is used to initialize a git repository.

Command:

```
./git init
```

Working Logic:

It just creates an empty folder named “.git” in the current directory by using mkdir system call. Also creates a version directory, log.txt, status.txt and add.txt.

2. Add: (Prakhar Gupta)

The git add command is used to add a single file or a single directory or all the contents of the current directory to the staging area of the git repository.

Command:
./git add <file name/directory name>

Working Logic:
The git add command adds a change in the working directory to the staging area. If the argument is a file, then it includes its name in add.txt. If the argument is a directory, it recursively saves all the files inside that directory in a vector and then it append all the file names to the add.txt file.

If the argument is "." it recursively finds and saves all the file names present in the directory and then appends it to the add.txt file.

Before writing to the add.txt file it first collects all the previously added file names in the add.txt file into a vector and then compares it to the newly added file names vector and this is how it makes sure that one file is only added once in the staged area.

Initial Checks:

- Check if the file exists or not.

3. Commit: (Karan Bhatt)

The git commit command saves the snapshot of the project's currently staged changes. Commits can be thought of as checkpoints. It creates a new directory on every commit named "v_n" where n is the number of the latest commit.

Command:
./git commit

Working logic:
The commit command creates a new version directory and copies all the files from the previous version directory into the new directory. It keeps track of

new/modified files in the latest version and adds/updates it to the latest version directory. At last, updates the log file.

Initial Checks:

- Check if the init command is executed by checking if the “.git” directory exists or not.
- There must be atleast one new/updated file in staging area for commit.

4. Status: (Aryan Gupta)

The status command shows current state of the working directory and staging area.

Command:

```
./git status
```

Working logic:

It records all the files in current working set and reads add.txt and status.txt files. When commit is performed files name with calculated SHA1 also added to status.txt.

It then compares list of all files with the files in add.txt and status.txt. If a file is not in add.txt and status.txt that means it is untracked file. If file is in add.txt then it is a new file and staged. If the file is in status.txt then it is modified file. Displays all Untracked, Staged and Modified files.

5. Rollback: (Anurag Ghosh)

The git rollback command is used to go back to previous versions of commit of the project and restore the specified state. It is primarily used with the intention of undoing changes to a repository's commit history.

Command:

```
./git rollback <mode> <checkpoint>
```

Working Logic:

The .git hidden folder made upon executing the init functionality, creates a subfolder called version. Upon a commit, a new subfolder under ‘version’ is created every time with the nomenclature ‘v_n’ where n denotes the version number. The version contains all the files pertaining to the respective commit.

Upon executing the rollback function, the intention of the user is to go back to a previous state of the project. Upon execution of the function, the version folder inside .git is opened up and the folder names are stored in a vector (v_1,v_2,..,v_n). The last version is taken to be the version to be deleted and the version latest to it needs to be restored. The contents of the program execution directory are also stored in a vector in order to restore the files. The last version folder is deleted and the latest version folder respective to it is compared with the files in the current directory. In case any folder/file has been deleted or modified, it is restored back to its previous state and all the changes are rolled back. The current directory state pertaining to a certain commit is achieved.

Special Modes:

1. **Command with no arguments:**

   If no arguments are passed, it is assumed that the user wants to rollback to the latest commit. In that case, the last stored commit version folder is deleted and the state of the current directory is restored to the commit state previous to it.
   To execute : **./git rollback**

2. **Command with checkpoint argument:**

   The user can execute the rollback command with a special argument by specifying a checkpoint, which means the number of versions the user wants the rollback to hop back to. For example, the user can specify the rollback to take the project to 2 commits back and restore that specific state. So in that case instead of running the rollback command twice, they can specify this request with the checkpoint argument.
   To execute: **./git rollback -c <number of hops>**

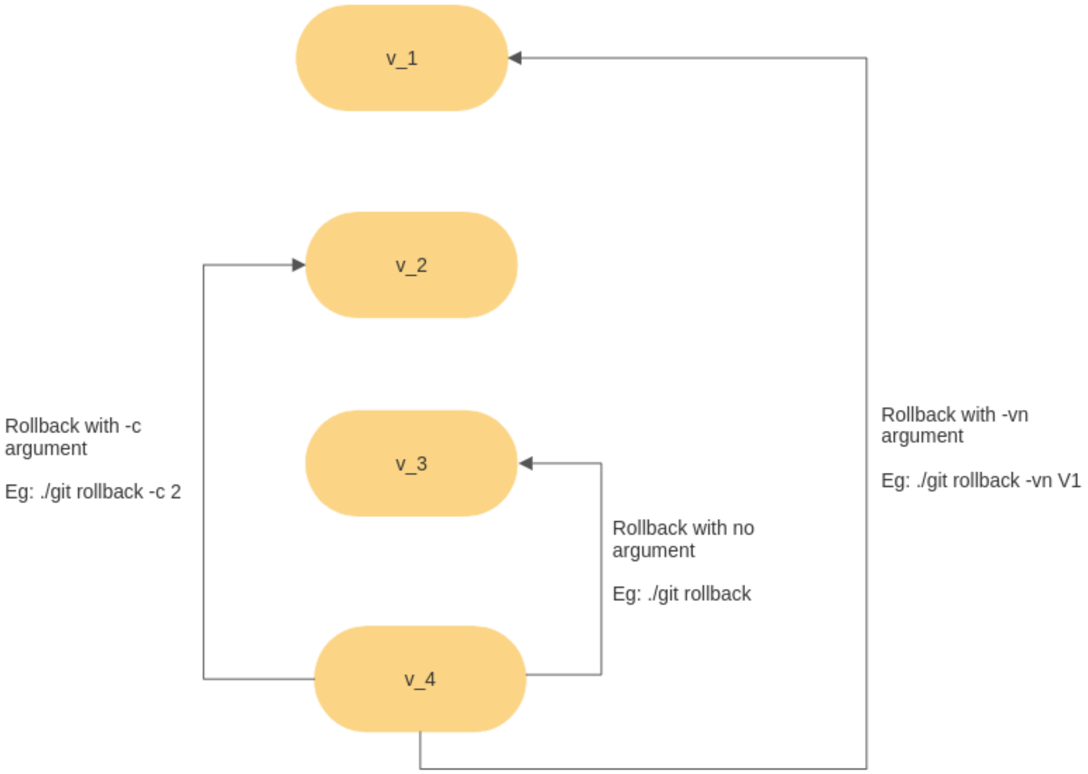
Example : **./git rollback -c 3** (for example: this will take v_5 to v_2)

3. **Command with version specific rollback:**

   The user can execute the rollback command with a special argument by specifying the name of the version to rollback to, in the format : **V<version number>.** Example : **V3**. This will take the current version of the folder to the state of the third commit.
   To execute : **./git rollback -vn V<version number>**
   Example : **./git rollback -vn V4** (This will take the folder to the commit state of version 4)

Initial Checks:

- If the git version folder has no commits, the rollback function cannot be executed and will display a custom message
- If the rollback command is executed with no arguments, the folder is rolled back to its previous state
- If the rollback command is used with the special arguments, the validity of the rollback is checked by comparing the number of commit folders in the version directory with the user input. In case of a logically invalid rollback request, a custom message is displayed on the screen and the operation is aborted.
- In case the version folder is deleted, the execution of the rollback function is stopped and the user is prompted to run the init command again to restore the folder.


v_1
v_2
v_3
v_4
Rollback with -c argument
Eg: ./git rollback -c 2
Rollback with no argument
Eg: ./git rollback
Rollback with -vn argument
Eg: ./git rollback -vn V1

6. Log: (Aryan Gupta)

The log command shows the history of all the commits and rollbacks done in the directory. It contains the version_number, message and timestamp of all the commits.

Command:
./git log

Working logic:
Whenever any commit or rollback is made, its details are appended in the log.txt file. So, when the git log command is run, it reads the log.txt file and shows all the commit/rollback history along with all its details.

7. Tracker Logs: (Anurag Ghosh)

It keeps logs of all the functionalities user has run into a file “trackerlog.txt”. The logs can be used to understand the flow of the program and further debug or track any problems.