INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

Project Report

On

Version Control System

SUBMITTED BY:

Peeyush Sahu	**2021202015**
Karan Negi	**2021201039**
Beeraka Krupa Kiranmai	**2021201022**
Jatin Gupta	**2021201048**

Guided By: Rishabh Malik (Teaching Assistant)
Submitted to: Manish Shrivastav

DECLARATION

We hereby declare that the project entitled “Version Control System” which is being submitted for the project of 1st semester at International Institute of Information Technology is an authentic record of our genuine work done under the guidance of Rishabh Malik, Teaching Assistant, International Institute of Information Technology

Date: 27/11/2021

Submitted by:
Peeyush Sahu
Karan Negi
Beeraka Krupa Kiranmai
Jatin Gupta

CERTIFICATE

This is to certify that the project report entitled “Version Control System” submitted by **Peeyush Sahu, Karan Negi, Beeraka Krupa Kiranmai, Jatin Gupta** has been carried out under my guidance and supervision. The project report is approved for submission requirements for the project in the 1st semester in the International Institute of Information Technology, HYDERABAD.

Date: 27/11/2021 **Prof: Manish Shrivastav**

VERSION CONTROL SYSTEM

Introduction:

What is “version control”, and why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For example, you will use software source code as the files being version controlled, though in reality, you can do this with nearly any type of file on a computer.

If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead. Version control systems are a category of software tools that help a software team manage changes to source code over time.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Developing software without using version control is risky, like not having backups. Version control can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers.

Problem Description:

Implement a git-like client system with limited capabilities. The program should be able

- To create/initialize a git repository, create a .git folder with the necessary details.
- To be able to add files to index and commit, maintaining the commit-ids so that retrieving them back could be done.
- To be able to see the status, diff, checkout previous commits, as shown by the git utility.

Solution Approach:

Our task is to keep track of various versions in our project in such a way that our solution is memory efficient. For that rather than storing the complete file in a separate version, we are storing the changes that were made on the file with respect to the previous version of that file. Using this approach we can save a lot of space.

Suppose we have a file of 100 MB and we made some changes to it. As per the traditional approach, we will store the complete file of 100 MB in our version folder. For every minute change, we are occupying unnecessary space. In our proposed solution rather than storing the complete file we are only storing the changes that were done on that file.

In order to get the previous version of that file, we will simply undo those changes which were present in our current version file. By using this approach we have reduced memory usage to a great extent. Similarly, we are storing only these changes w.r.t. previous version file in subsequent versions.

Implementation of our proposed solution:

Steps to install Team Rocket - Version Control System

1. Give executable permission to the file ‘install.sh’ by using command: ‘chmod +x path/to/install.sh’
2. Execute the ‘install.sh’ in superuser mode : ‘sudo ./path/to/install.sh’
3. After trgit is installed successfully, restart your terminal session for the trgit command to work.
4. Congratulations Now you can use the trgit (Team Rocket - Version Control System) from anywhere in your system.

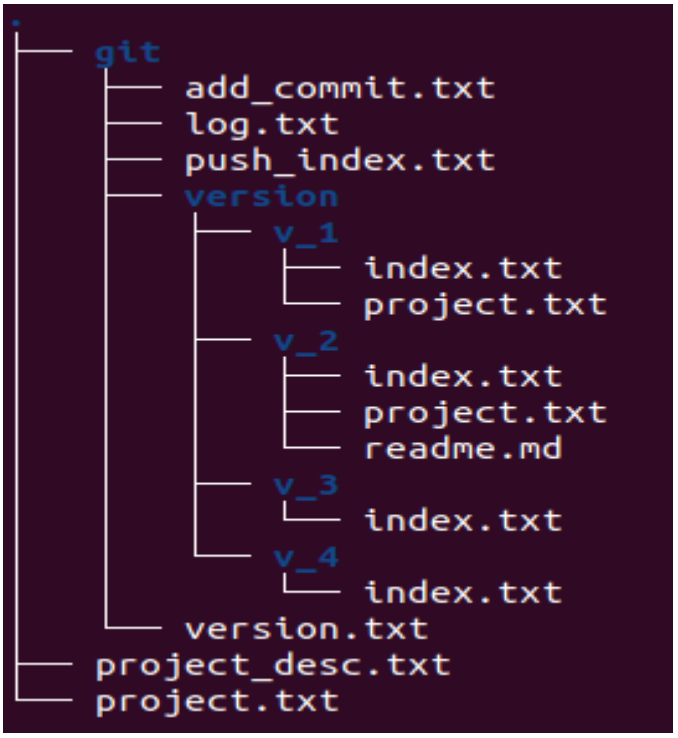

Fig1.1: Directory Structure

Commands Implemented

1. Init
2. Status
3. Add
4. Commit
5. Log
6. Diff
7. Push
8. Clone
9. Rollback
10. Pull
11. Create_repo

1. Init:

COMMAND NAME: init
COMMAND SYNTAX: trgit init

The init command creates a new repository. It can be used to convert an existing, unversioned project to a version control repository or initialize a new, empty repository. Most other version control commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.
It creates the “git” folder if not already present. The git directory contains :

- It contains a version directory with v_1 as a subdirectory which contains an empty index file that maintains the data for the respective version.
- version.txt: This file stores the version we are currently working on. It contains the initial value as 1.
- log.txt: It stores logs of all the commits of the version. Initially, it is empty.
- push_index.txt: This will store the recently pushed filenames and their corresponding SHA.
- add_commit.txt: It will store 2 bits and a remote repository path.One to check whether the ‘add’ command is executed or not. Another one is to check whether the ‘commit’ command is executed or not.

2. Status:

COMMAND NAME: status
COMMAND SYNTAX: trgit status

It gives the status of all the files.

- If a file is changed then it is shown in yellow color with the modified label.
- If the file is untracked then it is shown in red color with the untracked label.
- Else it is shown in green color with the tracked label.

working of status command :
In the status command, we are checking whether the file is already present in the previous version, if it is present then we are checking whether the SHA of the current file and file in the previous version is the same or not.

- If sha is the same, then the file is not changed and it will be labeled as tracked.
- If sha is not the same, then the file is changed and it will be labeled as modified
- If the file is introduced in the current version, then that file will be labeled as untracked. It means that the entry of that file is not present in our index file.

3. Add:

COMMAND NAME: add
COMMAND SYNTAX: trgit add <parameters>
COMMAND PARAMETER: <filename> / .

- add all files
    - By using the command (add .) all the files that are present in our current working directory (where our repo is initialized) will be marked as tracked and will move to the staging area.
- add a file by filename
    - By using the command(add <filename>), the file provided in the argument will be marked as tracked and will move to the staging area if not already.

working of add command:
In order to add file/files to our staging area, we are calculating the SHA of each file.

- If the sha of the file matches with the sha of the file in the previous version (which means the file is not changed) we will move to the next file and leave that file as it is.
- If both the sha doesn't match we are finding the diff between two files(the current version and the previous version) and storing that difference in a file and that file is stored inside curr version directory which is present in the "git/version".
- If the file was not present in the previous version and introduced in the current version, we are storing the complete file inside the current version directory which is present in "git/version".

4. Commit:

COMMAND NAME: commit
COMMAND SYNTAX: trgit commit

In commit, we are updating the current version in the "version.txt" file present in the git directory and updating the logs which contain the commit number, date & time of a commit, and the path of the directory (It represents the local repo) from where the commit was done.

5.Log:

COMMAND NAME: log
COMMAND SYNTAX: trgit log

This displays the list of all the commits along with commit number, date & time of a commit, and the path of the directory (It represents the local repo) from where the commit was done. All the information stored in log.txt is read and we display that information to the terminal.

6.Diff:

COMMAND NAME: diff
COMMAND SYNTAX: trgit diff <parameters>
COMMAND PARAMETER: <filename> / .

- diff all
    - By using the command(diff .), what we are doing is we are iterating to each file present in our current directory, if that file is present in our previous version also we are calculating the difference between the two files and showing the output to the user.
- diff by filename
    - By using the command (diff <filename>), what we are doing is if the file provided in the argument is present in our previous version also we are calculating the difference between the two files and showing the output to the user.

7. Push:

COMMAND NAME: push
COMMAND SYNTAX: trgit push

This command will push the changes to the remote repo.

working of push command:
We are iterating over the entries present in the index file and creating a map that contains the filename as key and SHA of the file as value. Similarly, we are creating a map for the remote repository by iterating through the push_index.txt file present in the local repository. Now we are comparing both maps. We have three cases in it:

- Different SHA for same files: This case represents that the file is modified in the local repository. So we are updating the file in the remote repository with local changes.
- File present in the local repo but not in the remote repo: In this case, we are simply copying the file to the remote repo.
- File present in the remote repo and deleted in the local repo: We are deleting it from the remote repository also.

8.Clone:

COMMAND NAME: clone
COMMAND SYNTAX: trgit clone <parameter>
COMMAND PARAMETER: ./path/to/remote-repository

This command is to create a copy of the remote repo in our local directory. By using this command the local repo will be initialized and the content will be updated according to the remote repo.

9.RollBack:

COMMAND NAME: rollback
COMMAND SYNTAX: trgit rollback <parameter>
COMMAND PARAMETER: <version_number>

This command is used to get the changes that were done in the previous commits. We are recreating the file (as it looks in the specified version) and putting that file to our new version. One thing to note here is it is not removing the current version, We are creating a new version that is identical to the required version which is important for the integrity of your revision history and for reliable collaboration.

10. Pull:

COMMAND NAME: pull
COMMAND SYNTAX: trgit pull

This command will fetch the changes from the remote repo.

working of pull command:
We are iterating over the entries present in the push_index.txt file in the remote repository and creating a map that contains the filename as key and SHA of the file as value. Similarly, we are creating a map for the local repository by iterating through all the files in the local repository and getting their SHA. Now we are comparing both maps. We have two cases in it:

- Different SHA of the same file: This case represents that the file is modified in the remote repository. So we are updating the file in the local repository with remote changes.
- File present in remote but not in the local repo: In this case, we are simply copying the file to the remote repo.

11.Create_repo:

COMMAND NAME: create_repo
COMMAND SYNTAX: trgit create_repo <parameter>
COMMAND PARAMETER: ./path/to/create/remote-repository

This command will initialize the remote repo folder which will act as the remote repository.

Scope of Upgradation:

- Currently, this version control system works with files only. In the future, we can modify it for directories also.

- We have implemented the subsets features of currently used version control systems like Git, Concurrent Version System, Apache Subversion, etc.

- Currently, our remote repository is present in our local machine only, we can add support to connect it with the remote machine using socket programming.

Conclusion:

So, we have successfully created the version management system with C++ technology. We have included basic operations like init, status, add, diff, commit, rollback,log,clone, push,pull and create_repo.