Team Name: Kernel Panic

Version Control System

27th November 2021

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Submitted To:

Dr Manish Shrivastava

Assistant Professor

Submitted By:

Abhishek Bisht (2021202014)

Kshitij Gupta (Roll No. 2021201075)

Sarthak Rawat (2021202006)

Shreyash Agrawal (2021201074)

Introduction

A version control system is a software which is mainly used to keep track of the history of a project. VCS, is a specialized software with the primary goal to manage & record the changes to codebases over time, a process called version control.

Version control systems allow multiple developers, designers, and team members to work together on the same project. These systems are critical to ensure everyone has access to the latest code. As development gets more complex, there's a bigger need to manage multiple versions of entire products.

Problem Description

To Design the system which will provide functionalities similar to advanced version control systems like git. The system should allow the user following features:

- It should keep a track of the code changes
- It should allow the user to make checkpoints of such changes within his code which he can view or revert back completely.
- It should be able to save the code in a different location, and allow it to be fetched.

However, most modern version control systems have many additional functionalities as well. In this project we aim to implement a subset of those functionalities, few of which are listed below along with their description.

Commands:

1. **Init** - This command will initialize a directory to put it under version control. It will create a directory named '.vcs' inside the directory from which the command was executed. It will contain all the relevant data that is necessary for keeping track of history of the project as well as other functionalities.

2. **Add** - This command will add the modified file that is passed to it as an argument to a staging area. Staging area is just an index file that keeps a list of files that are currently being tracked by the version control system.
3. **Commit** - This command will take the newly added tree along with author name, commiter name and commit message; and create a new commit object. This object will be stored in the .vcs/obj directory. The current branch will now contain the hash of this newly created commit.
4. **Diff** - This commands takes the file as an input and prints the difference between the lines which are changed before running add command.
5. **Pull** - This command will pull a working vcs repository and run checkout on the latest commit available. Only local repositories are implemented for pull.
6. **Push** - This command will push the current working vcs repo to a local folder.
7. **Checkout** - This command will checkout a particular commit or a branch and will modify existing files to that commit/ branch.
8. **Rollback** - This command will rollback to the previously done commit removing all changes to the staging area.
9. **Log** - Shows logs of commits done and current head position.
10. **Status** - It gives the status of all the files. It keeps track of Files which are modified, untracked and tracked but not modified.

The programming language that we will be using is Python.

Solution Approach

In our solution there are three main objects each of which has been described below.

1. **Blob:** Every file that has been stored in the version control has been stored in the form of blob objects. A blob is just the compressed form of the file contents that is under version control as shown in Figure 1.

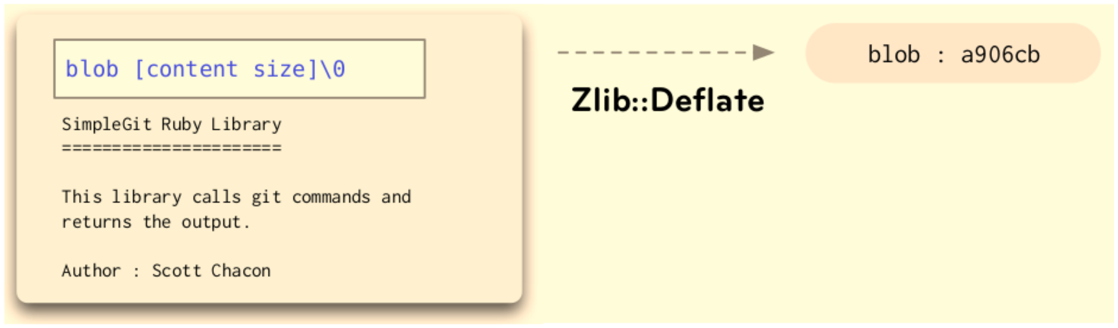
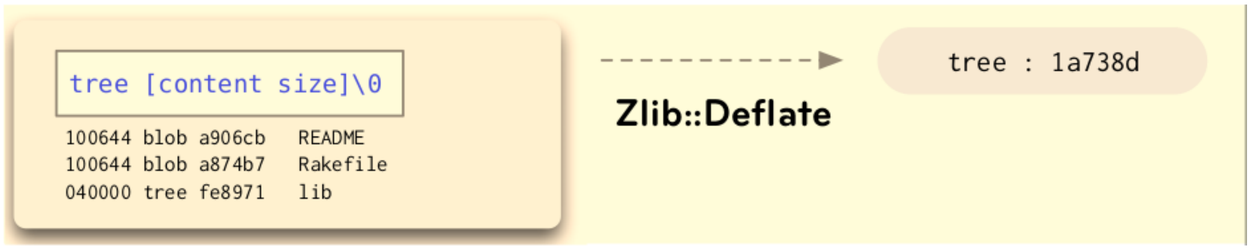


Figure 1.

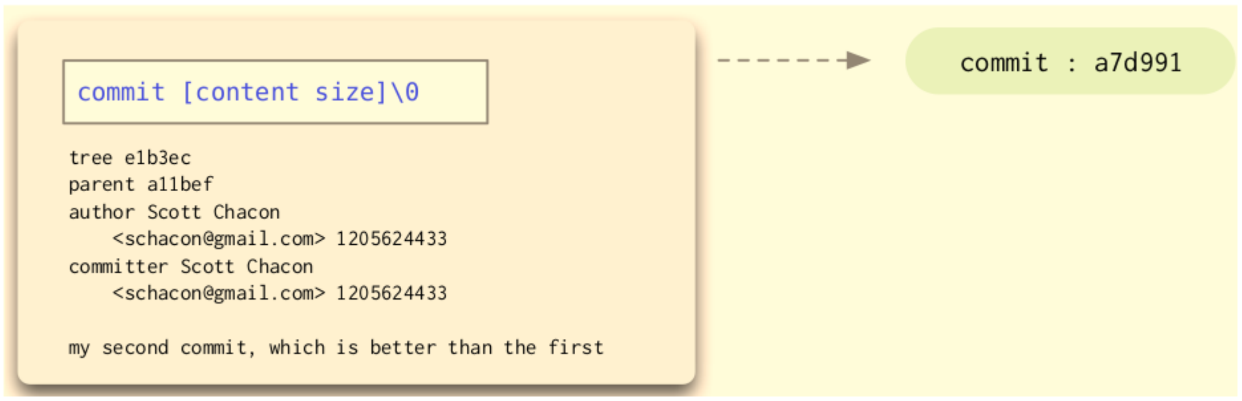
2. **Tree:** Every directory under the working dir is represented by a tree object. It is stored in compressed form by our version control system. A tree contains an entry for each file and directory inside the directory that it is representing. Figure 2 shows the visual representation of a tree.



```
tree [content size]\0
100644 blob a906cb    README
100644 blob a874b7    Rakefile
040000 tree fe8971    lib
```

Figure 2.

3. **Commit:** A commit object represents the snapshot of the entire working directory. It contains a reference to the tree object corresponding to the root of the working directory and a reference to the previous/parent commit if any. Figure 3. Shows the visual representation of a commit object.



```
commit [content size]\0

tree e1b3ec
parent a11bef
author Scott Chacon
    <schacon@gmail.com> 1205624433
committer Scott Chacon
    <schacon@gmail.com> 1205624433

my second commit, which is better than the first
```

Figure 3.

Storage of objects

Objects are stored in a directory with path '.vcs/objects' as files where each file is named as the hash of the corresponding object. Hash is calculated using the SHA256 algorithm.

Relation between blob, tree and commit objects:

Figure 4 shows a sample working directory and the corresponding commit objects.

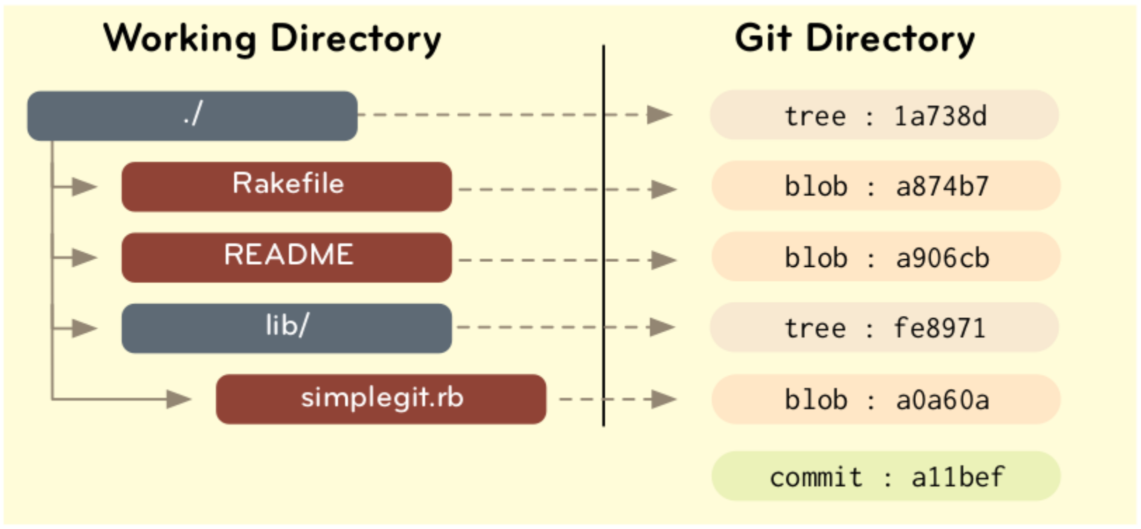


Figure 4.

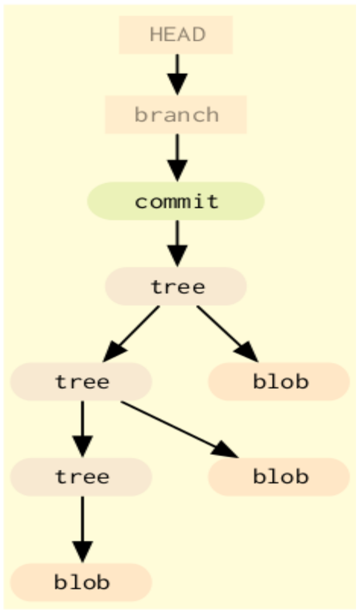


Figure 5. shows the link between the three types of objects listed above. The commit object has a reference to the tree object corresponding to the root of the working directory. The tree in turn has entries for other tree objects and blobs. Branch is a file that stores the hash of the commit. Head in turn stores the name of the branch.

Figure 5.

Staging Area:

Staging area is basically a file that keeps track of all the files in the working directory that are under version control. In our implementation the path of the index file is '.vcs/index'.

When a file is added to the version control system, it is checked for existing entries in the index file. If there is already an entry, the current file's hash is compared with that in the entry. If the hash is different then a new blob object is created for the contents of the new file and stored in the objects directory with its name as hash.

Creating of new Commit:

When the commit command is executed, the system scans the index file for any new modifications. If it finds new or modified entries, it traverses the current commit tree and clones all the objects that have been modified by this new entry.

Work Distribution

Sarthak Rawat

- Commit Utility - Create a new commit of the newly created tree. Update the current branch.
- Log - Display all commits from the current commit to the oldest commit.
- Diff - Display differences between an indexed file and a given file.
- Status - Display status of all the files in the working directory. The file can be tracked, untracked or modified.

Shreyash Agrawal

Developed the main controller which calls all other commands and commit tree utility that is responsible for creating the commit tree object which does the following: -

- All the directories in the project are converted to form a hierarchical structure i.e. a tree.
- Recording all the changes done in a commit at each directory level in the tree object
- Taking all the staged/tracked changes from the staging area, and then recursively creating the tree objects for each directory in the project.
- Creating a new tree object after each commit
- The new tree object would take in the old tree and only add/replace/modify the staged changes in this tree and form a new one.

Kshitij Gupta

Created utility functions and push, pull and rollback commands:

- The following utility functions were created in the util.py file:

- *compress*: takes paths of files/dictionary objects and stores it as a binary compressed form in .vcs/objects directory. Also returns the SHA256 of the original file/tree.
    - *decompress*: Given a SHA256 value, it fetches the binary objects stored in .vcs/objects directory and decompresses into a file or a tree object.
    - *Getting SHA*: calculates the SHA256 of a file, a tree or a commit object. Also used
    - *Updating HEAD pointer*: get/set the HEAD pointer with the branch name or the commit hash
    - *Last commit hash*: returns the commit hash of the last commit done of that branch
    - *Getting modified entries*: returns the list of entries in the index file that are modified or newly added. Mostly used by the add command.
    - *Setting the modified status* of the files in the Index file
- **Push**: Implemented the functionality as python script, push command will take a folder (remote repo) as an input and will copy the contents in the .vcs folder to the remote repo. After copying, it will make the folders structure from the latest commit.
- **Pull**: Implemented the functionality as a python script, pull will take a folder (remote repo) as an input and will copy the .vcs folder to the current working directory. It will checkout the latest commit after copying.
- **Rollback**: Implemented the functionality as a python script. will rollback all the changes made before add is being run.

Abhishek Bisht

- Implemented ‘init’, ‘add’ and ‘checkout’ functionality.
- **init:** implemented the functionality as python script which created a .vcs folder inside the directory from where it is executed.
- **add**: implemented the functionality as a python script.
    - Takes the file/directory path which is to be added to version control as input.

    - Create an index file if it is the first execution of add and adds the file/directory to the index file.
    - If the index file already exists, it checks for the entry corresponding to the input file/directory and checks for any modification using SHA.
    - Add/Update the entry in the index file if there is a modification.
- **checkout:**
    - Script takes a commit hash as input.
    - Loads the snapshot corresponding to the commit
- **Staging area:** Implemented staging area in the form of index file which keeps a track of all the files under version control.

Problems Faced and Shortcomings

Problems faced

These were the few problems we faced while working on our project but all of them have been resolved by the end:

- Tree structure formation: the trees were not formed properly
- File path problems (Absolute and Relative)
- HEAD pointer was not getting updated
- Index file was making duplicate entries for same file
- .vcs folder was getting included in the add function

Shortcomings

There are a few shortcomings in our current implementation of the Version Control System:

- Deleted files: Our implementation does not track the deleted files.
- Branching and Merging: We have not implemented the branching and merging functionalities like a real git version control system.

Learnings

Python

Working with Python libraries such as ZLib, HashLib, Shelve, Pickle

Collaboration

This project gave us the opportunity to learn a lot about how to manage a project. Creating a project with collaboration with others is quite a difficult task. We used github to streamline our workflow.

VCS Architecture

Studied the Internal structure used by popular VCS providers, such as GitHub

Result

We have made a working application that can be used by a programmer to manage their project. They can use this application to manage their codebase to their preference.

Conclusion

We developed the application which serves as a means to manage the Client's project files in a more efficient manner. It provides the user with features like version History, code backup, and more.

References

https://github.com/pluralsight/git-internals-pdf/releases