LINUXTERS

VERSION CONTROL SYSTEM

- Sumeet Agrawal
- Ashish Chauhan
- Gargi Dwivedi
- Rishu Toshniwal

In case of fire -> git commit, git push, escape

Introduction:

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.

Git is the most popular software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Git was created by *Linus Torvalds* in 2005, for development of the Linux kernel with other kernel developers contributing to its initial development.

Git vs. Github:

Both Git and GitHub give programmers valuable version-control functionality so that they can build ongoing coding projects without being afraid of messing everything up. GitHub just takes things a little bit further than Git, offering more functionality and resources, as well as a place online to store and collaborate on projects.

The difference? Simply put, Git is a version control system that lets you manage and keep track of your source code history. GitHub is a cloud-based hosting service that lets you manage Git repositories. If you have open-source projects that use Git, then GitHub is designed to help you better manage them.

Problem Description:

Implement a git-like client system with limited capabilities. The program should be able to perform following functions:

• create/initialize a git repository, creating a .git folder with the necessary details.

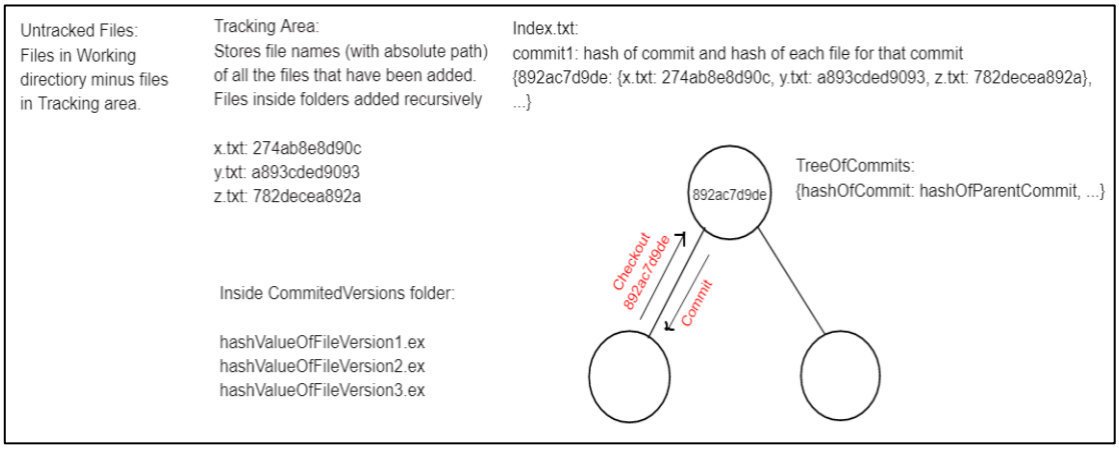
• add files to index and commit, maintaining the commit-ids so that retrieving them back could be done.

• see the status, diff, checkout previous commits, as shown by the Git utility.

Commands to be implemented:

1. init
2. add
3. status
4. commit
5. rollback
6. checkout
7. diff
8. log
9. push
10. pull

Solution Approach:



(I) *Design*

```
def __init__(self, path, force=False):
    self.gitdir = os.path.join(path, ".git")
    self.logfile = os.path.join(self.gitdir, "log.txt")
    self.gitRepoPath = os.path.join(self.gitdir, "Repository")
    self.trackedFilePath = os.path.join(self.gitdir, "trackedFile.txt")
    self.commitHeadPath = os.path.join(self.gitdir, "commitHead.txt")
    self.trackingAreaPath = os.path.join(self.gitdir, "trackingArea.json")
    self.treeOfCommitsPath = os.path.join(
        self.gitdir, "treeOfCommits.json")
    self.indexFilePath = os.path.join(self.gitdir, "index.json")
    self.workingDirectoryFiles = set()
    self.modifiedFiles = set()
    self.trackedFiles = set()
    self.treeOfCommits = {}
    self.trackingArea = {}
    self.index = {}
    self.commitHead = None
    self.RemoteRepo = "/Users/ashishchauhan/Downloads/Remote"
```

(II) *Some important files and data structures*

- *trackingArea: dictionary*
    - Stores the files that have been git-added once and are being tracked as: <filename with absolute path> : <SHA digest of the recent version>.

- *trackedFiles: set*
    - Stores the names of files (with path) that have been added but not yet committed.

- *modifiedFiles: set*
    - Stores the names of files (with path) that have been modified since the last commit.

- *treeOfCommits: dictionary*
    - Stores the parent's commit ID corresponding to each commit ID

- *index: dictionary of dictionaries*
    - Stores the commit IDs as keys and corresponding dictionaries of <filename with absolute path> : <SHA (of respective version)> as values.

- *commitHead: string*
    - Stores ID of the recent commit.

Important modules:

1. git init: Creates the .git folder, with files shown above inside it.

2. git add: Adds the files/folders provided as arguments to the *trackingArea* dictionary along with its current SHA hash and filenames to the *trackedFiles* set.

3. git commit: Gets a commit ID from the timestamp, calculates and stores current SHA of files with their absolute paths in the *index* dictionary, empties the *trackedFiles* set. Copies the current file to *Repository* inside the .git folder as "hash.extension" and writes to *log*.

4. git status: Prints *trackedFiles*, gets the current working directory to print untracked files. Compares the current hash of the files in SHA hash of files in current commit (commitHead) to its hash in the parent commit and prints modified files.

5. git rollback: Gets the versions of files in the recent commit using the *index* dictionary and *commitHead.* Restore all the files from that commit.

6. git diff: For all files in the two commit ids provided, looks for added and deleted files using the *index* dictionary, and for modified files prints differences using the difflib module.

7. git log: Reads logs from *log* file and prints it to the screen.

8. git push: Move the current commit files to a remote repository. (local machine as of now)

9. git pull: Get files from remote repository to local repository.

Problems Faced and Shortcomings:

1. To get every team member on the same page so that we all can proceed in same direction.

2. Syncing in terms of time and with each other's code.

3. A lot of time was invested to make the code readable so that other group members can easily get the updates and contribute.

4. Thinking of a design and architecture which can implement all the required functionalities of the project was the most challenging but exciting task.

5. With so much going on, time management was always an issue.

Learning:

Apart from one of our teammates, who has experience in the industry, no one knew much about Git or Version Control System beforehand.

We started by giving ourselves some time to get acquainted with the purpose that Git serves, looking inside the .git directory from the other repos like SSD labs.

Links were shared around, and we put forward design proposals. We were unsure about the physical storage of files, storing files with the same name in a folder, how commits are linked, just the differences are stored or whole files, among many more.

After some research, we met on 17th November and finalized the high-level architecture. Meanwhile, we had started implementing little independent modules on our ends.

Funnily enough, we realized that not all of us were coding in Python. We might not have handled conflict resolution in version control, yet we mastered it in this project.

As the project developed, we started meeting more frequently, sharing every little development, such as better documentation or the addition of colors to the output (as can be seen by the commit messages on our Repository).

Thus, we can't name solid tasks implemented by a teammate, every one of us contributed somehow to all the modules.

Key takeaways: improved understanding of git, git log, checking out commits, resolving merge conflicts, Python, OOPS, Code Integration, Debugging and Testing.

Result:

Saving a version of the project after making changes is an important task. This task becomes tedious and quite confusing if people are not using VCS.

Every time people commit changes, they create a new version of the corresponding file .With the help of VCS that our team developed, software developers can request required version at any time, and they will have a snapshot of the complete project.

VCS allows the older version of the code to be safely stored in the VCS repository. It means people can always go back to a specific version of the file. VCS can be extremely useful because it will allow people to recover from accidental deletions, edits.

Conclusion:

Software developers should have a basic understanding of what Version Control System is and how to use it. The adoption of VCS is must in software development.

It helps software developers to manage their codes easily because it is common to have a lot of changes involving addition or deletion of features.

To adopt a VCS, a software developer must know different VCS commands and basic insights of how these commands work internally. This helps developers to understand the importance of each command and how it is used.

This project helped our team to get this basic understanding of VCS commands and their insights. We used Version Control over Github to create this project on Version Control.

This has been a great experience with loads of learning in every curve of software development.