INTERNATIONAL INSTITUE OF INFORMATION TECHNOLOGY HYDERABAD

PROJECT REPORT FOR OPERATING SYSTEMS

VERSION CONTROL SYSTEM

SESSION: JULY-2019 DEC-2019

GUIDED BY:

DR. SHATRUNJAY RAWAT

DANISH ZARGAR (TA)

SUBMITTED BY:

SHANU GANDHI 2019201014

SHUBHAM AGARWAL 2019201085

SHRAYANS JAIN 2019201086

YASH UPADHYAY 2019201098

INDEX

1.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 the 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.

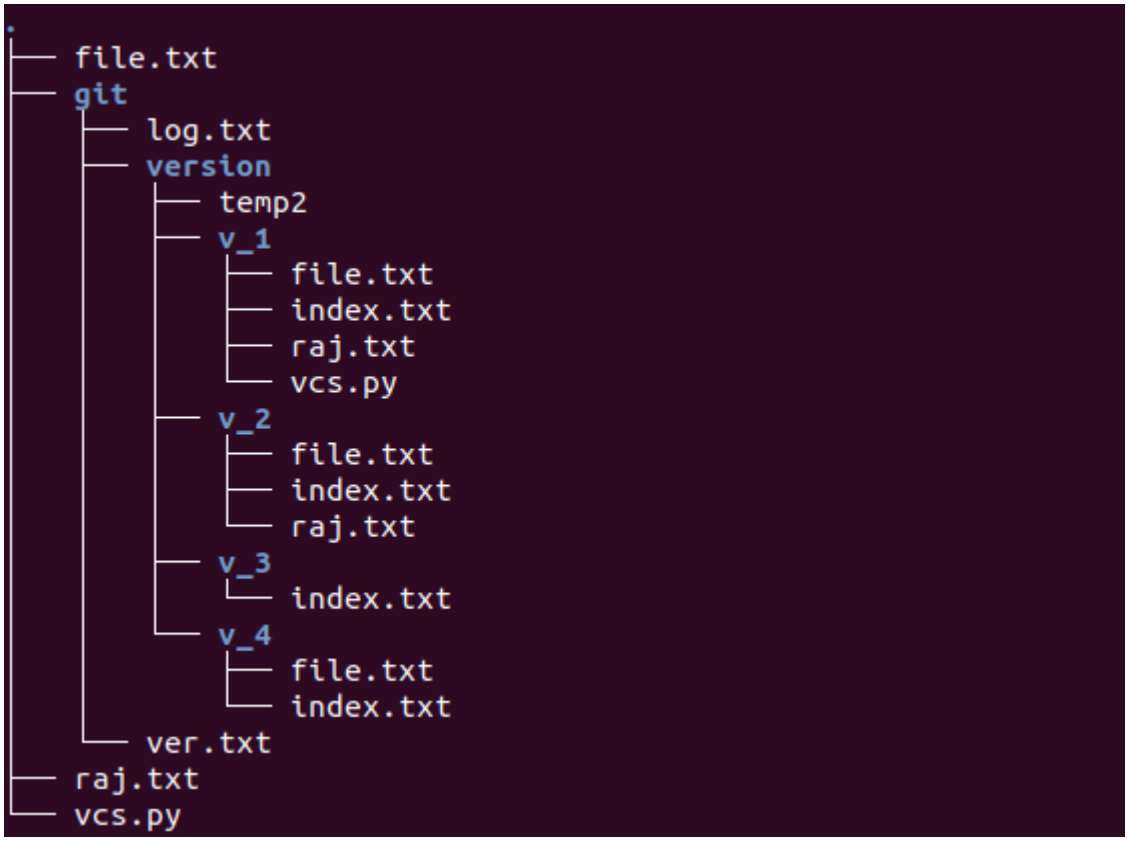
Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer 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.

2.ARCHITECTURE

Initially if we want our version control system in a directory than we need to do init in the directory. By init command we generate a folder called “git” which will maintain all the data of our versions control system. Initially it contains two files one for getting current version and a log file to contain the logs of commit. The version file initially contains entry as ”1” as current version and log file is empty. Also, it contains another directory which contains respective directories for all the versions created till now.

Following is the directory structure of Git Folder:



In version directory for each version we have an index file which store file name, hash of file, parent, modified and list of all the versions where the file was changed. This is used to get information of all the files in current version. Its entry format is as follows:

[Filename, SHA1 of file, Parent, Modified, Link-List]

The entries signify the following:

File name: the name of file.

SHA1 of file: this is used in status command i.e. if file is changed then its hash is also changed so we show the filename as red in status command output

Parent: This shows the parent version of file i.e. where the file was created.

Modified: This entry is “M” or ”O” i.e. if file is modified in current version its value is “M” and if its not its value is “O”. If in a version we have Modified bit as “M” and parent is also equal to current version than this version contain the full file else only diff is stored for files.

Link-List of modified versions: This list contains all the versions where the file is modified and hence the first entry signifies where the file was created. This is used to revert back and to check diff as we can get a file from going to its origin and patching diffs of all the versions where it was modified.

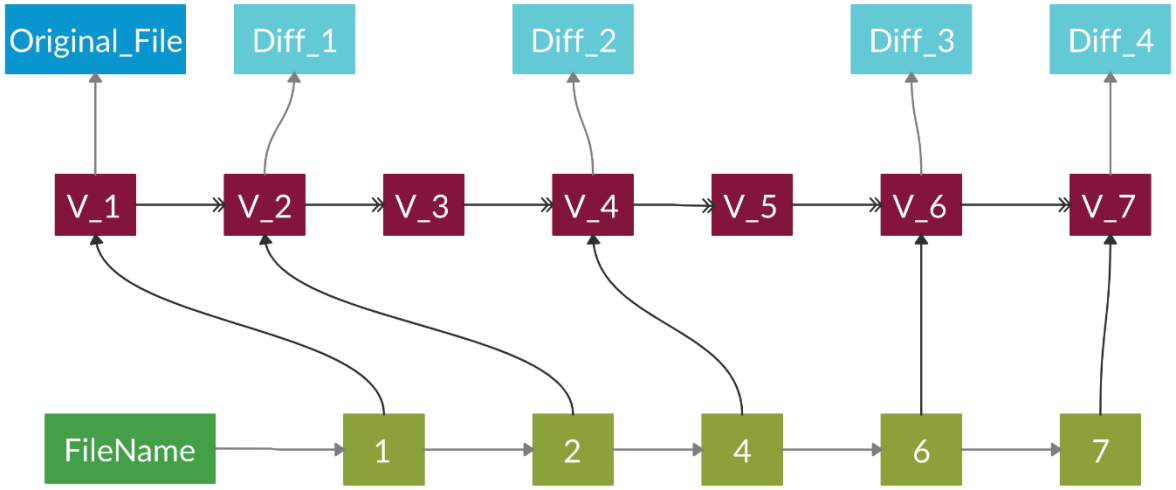
Following is the structure of Index File:

```
vcs.py 85367a7d2fad8d6aaef48765a05f2469ef137d95 1 0 1
raj.txt aac2ed58490c7a3f64ad7c503d19499e02021e9a 1 0 1 2
file.txt bc567e36785e16e8d24a70b8f80d7186ed6f1456 1 0 1 2 4
```

For **space optimization** we are only storing the diff of files in version folder if the file is changed and nothing is stored if the file is not changed. For getting diff in status command we goto initial file and patch cumulative diff of all respective versions where it was changed to get the file.

Now when we do “status” command we get to know all the changed and new added files in our directory. Then we do an add command it stores the difference of the files if changed or full file if file is added in current version. Next is when we commit, a new folder for next version is created and the other older versions folder freezes which can be used for rollback or checking status.

Following is the structure of Versions and diff:



Suppose we have a new file in version 1 and then we add it so whole file is added in version 1 folder also entry 1 is added in the link list of file in index file. Next we commit it so versions 2's folder is created(index file from version 1 is copied in it). Now we changed the file in version 2 and add it. So now only diff is stored in version 2 folder and entry '2' is added in link list . Next we commit it so folder for version 3 is created . We did not changed file in version 3 so nothing is added in folder and index is unchanged in version 3. Now after commit we changed in 4rth version so diff is stored in 4rth version and entry '4' is added in link list in index file. And this goes on.

Finally if we want to check status in v7 , we see link list of file .Thus we go to v1 take file and patch successive diffs from v2 v4 and v6 to create a temporary file and finally apply diff command to find diff of current file with the temporary file.

3.HOW TO RUN

We need to first place our Python files in the folder where we want to create our versioning system. Then we need to run our code by using terminal as

./version_control.py command

Where command is the command which we want to execute

Following are the commands that we have created:

1.Init

2.Status

3.Add

4.Commit

5.Rollback

6.Retrieve

7.Log

4.COMMANDS

Init

1.COMMAND NAME: Init

2.COMMAND SYNTAX: ./version_control.py init

3.COMMAND PARAMETER: It doesn't require any parameter

4.WORKING: It creates the "git" folder if not. Git directory contain ver.txt for which tells on which version we are working. File ver.txt contain the initial value as 1. In log.txt we maintain all the log of all the commits of version. Initially it is empty. It also have version directory with v_1 as subdirectory have the empty index file which maintain the data for respective version.

Status

1.COMMAND NAME: status

2.COMMAND SYNTAX: ./version_control.py status

3.COMMAND PARAMETER: It doesn't require any parameter

4.WORKING: It gives status of all the files. That is if a file is changed than it is shown in red colour else it is shown in green colour. This is done by using index file that is if a file is created in same version it will be red and if it is changed from previous version than it creates a temporary file till previous version by getting link list from index file and checks differences between temp file and current file by comparing SHA1.

Add

1.COMMAND NAME: Add

2.COMMAND SYNTAX: ./version_control.py add

3.COMMAND PARAMETER: We can give filename to add or nothing.

4.WORKING: If we give filename then it will only add that file to the next commit and if we give nothing then all files which are changed are added to next commit. Also it adds the whole file in current version folder if the file is created in the same version or else it only stores the diff of the file from previous version.

For patching a file from a diff which is stored in various version we use a linked list type structure which is stored in index file and recursively patch the file by considering two diff file at a time. After producing a desired file from recursive patch we produce a diff from that file and a file in current working directory and store that diff in that version directory on which we are working.

Commit

1.COMMAND NAME: commit

2.COMMAND SYNTAX: ./version_control.py commit

3.COMMAND PARAMETER: It doesn't require any parameter

4.WORKING: It commits the current version that is frezzes the current version folder. create a new version folder and increment the current version. After commiting a version we need to enter its details that is version number and date and time in log file.

Rollback

1.COMMAND NAME: Rollback

2.COMMAND SYNTAX: ./version_control.py rollback

3.COMMAND PARAMETER: It doesn't require any parameter

4.WORKING: It rollbacks to the previous version. That is if we are in a version x+1 and we have added or updated some changes than it replaces file files changed till last commit. That is undo changes ttill last successful commit.It also uses index file to do so . That is it reads the index file than creates all the files till the last commit and then delete all files and stores the temp created file in current working dfirecytory. Also it delete the current versions folder and decrement the version number.

Retrieve

1.COMMAND NAME: Retrieve

1.1.COMMAND SYNTAX: ./version_control.py retrieve -a VersionNo.

1.1A.COMMAND PARAMETER: "-a means all" and "version_number"

1.1B.WORKING: It will retrieve all files from passed version no. folder after patching . For patching it will take index file of passed version no. folder. All the patched file will saved in retrieve_version_number folder.

Note in the whole process we won't be deleting any versions like rollback.

1.2.COMMAND SYNTAX./version_control.py "retrieve sha of file" " VersionNo."

1.2A.COMMAND PARAMETER:"sha of file we want to retrieve" and "version_number"

1.2B.WORKING: It will retrieve file having sha passed as argument. It will locate file through index file entries then it will patch the file. After patching it will save file in the retrieve folder of of version number passed. Here also it won't delete anything through the process.

Log

1.COMMAND NAME: log

2.COMMAND SYNTAX: ./version_control.py log

3.COMMAND PARAMETER: It doesn't require any parameter

4.WORKING: We read the log file stored and display it. It contains all the details of all commits. That is commit number and date and time of commit.

Diff

1.COMMAND NAME: diff

2.COMMAND SYNTAX: ./version_control.py “diff” (.) / (filename)

3.COMMAND PARAMETER: . for all files and we can pass filename also.

4.WORKING: It will read index of file and based on list of versions we are creating last commited temp file and then we will find diff between temp and original(current) file. And the output if exist will be displayed on terminal as per the command.

4.CONCLUSION

So, we have successfully created the version management system with python technology. We have included basic operations like **init, status, add, diff, commit, rollback, retrieve** and **push** (in client server socket programming manner).

5.REFERENCES and GUIDANCE

1. https://www.atlassian.com/git/tutorials/what-is-version-control
2. https://en.wikipedia.org/wiki/Version_control
3. https://wyag.thb.lt/