Operating Systems (OS)

<u>Git</u>

<u>Project Report</u>

By Team: “The_running_threads”

<u>Team Members:</u>
Praveena Sidgar – 2019201042
Nabhiraj Jain-- 2019201062
Vikram Keswani – 2019201059
Shraddha-2019201001

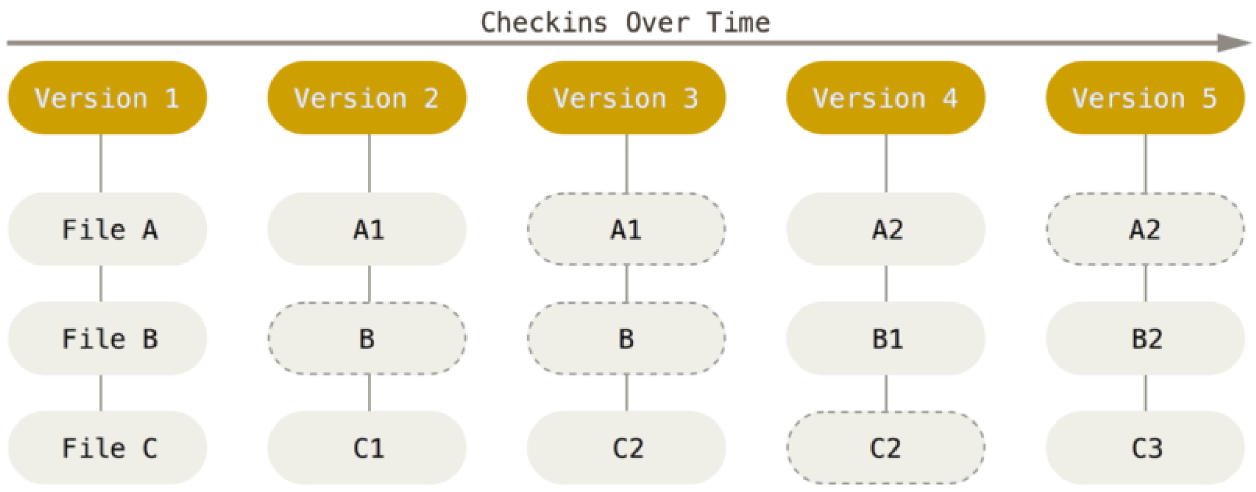
PROJECT OBJECTIVE:

1. Project Statement:-
* Version Control System (git).
*Implement a git like client system with limited capabilities.
*The program should be able
- To create/initialize a git repository
  creating 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.could be done.

2.To learn various functionalities performed by git and implement them in our own way such as:
- Init
- Add
- Commit
- Status
- Revert
- Push
- Pull
- Log
- Checkpointing

How Git Works?

Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a stream of snapshots.



- **Git Has Integrity**
  - ➔ Everything in Git is checksummed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it.
  - ➔ The mechanism that Git uses for this checksumming is called a SHA-1 hash.

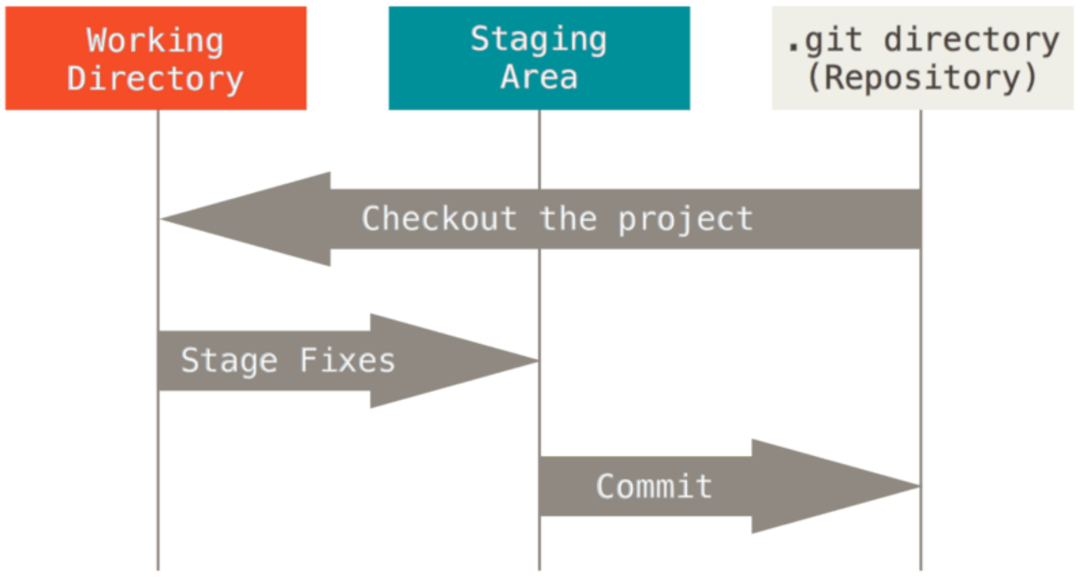
- **Git Generally Only Adds Data**
  - ➔ When you do actions in Git, nearly all of them only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way. As with any VCS, you can lose or mess up changes you haven't

committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

- **The Three States:**

Git has three main states that your files can reside in: modified, staged, and committed:

- ➔ **Modified** means that you have changed the file but have not committed it to your database yet.
- ➔ **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- ➔ **Committed** means that the data is safely stored in your local database.



- The basic Git workflow goes something like this:
    1. You modify files in your working tree.
    2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
    3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

“If a particular version of a file is in the Git directory, it’s considered ***committed***. If it has been modified and was added to the staging area, it is ***staged***. And if it was changed since it was checked out but has not been staged, it is ***modified.***”

How is our git different from this?

- Just storing the difference instead of the whole file in every successive commit so that it is space optimized.
- MD5 used for calculating hash value instead of SHA1.
- Platform Independent because it is coded in python.

IMPLEMENTATION:

Basic Concept-

- The history of the complete project is maintained using the successive difference file created at each commit.
- All the required metadata is present in .mygit folder
- The directory structure is as follows.
    - .mygit
        - master_file.txt
        - log
        - commit_<integer index>
        - diff<integer index>
        - tempdiff<integer index>
        - diff_counter
        - add_diff.txt
        - addlist_unique.txt
        - myfile_counter.txt
        - glob_empty (empty file)

Each of these file store some data which are necessary for safe project storage and retrieval.

master_file.txt: stores a number showing how many commits are done till now.

log: maintains the log of each commit with respective comment.

commit_<integer index>: stores all the meta information for commit number <integer index>

diff<integer index>: stores the difference of the file from previous commit which file and which commit can be mapped by the help of commit_<integer index> file.

tempdiff<integer index>: stores the same information as diff<integer index> but tempdiff<integer index> is temporary and is for add file unlike diff which is fore commited file.

diff_counter and add_diff: contains the value of next <integer index> for diff and tempdiff files respectively.

addlist_unique.txt: contains the metadata of all the files which are added but not yet committed.

myfile_counter.txt: stores number of file added till now (in addlist_unique.txt).

End User Documentation:

1.To initialise the repository:
- ➔ ./my_pure_back.py init

2.To check status:
- ➔ ./my_pure_back.py status

3.To add files:
- ➔ ./my_pure_back.py add #file_name1 #file_name2...

4.To commit:
- ➔ ./my_pure_back.py commit "#comment"

5.For checkpointing:
- ➔ ./my_pure_back.py checkpoint #commit_number

6.To check log:
- ➔ ./my_pure_back.py log

7.To revert:
- ➔ ./my_pure_back.py revert

8.To push:
- ➔ ./my_pure_back.py push

9.To pull:
- ➔ ./my_pure_back.py pull