INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

IIIT Hyderabad

PROJECT REPORT
ON

Version Control System (VJAY)

SUBMITTED BY: –

- Vishal Malhotra **2019201018**
- Yallamanda Rao Mundru **2019201029**
- Anurag Pateriya **2019201057**
- Jeevesh Kataria **2019201058**

Guided By: **Danish Zargar (Teaching Assistant)**

Submitted to : **Shatrunjay Rawat**

DECLARATION

We hereby declare that the project entitled **"Version Control System"** which is being submitted for project of 1st semester in **International Institute of Information Technology** is an authentic record of our genuine work done under the guidance of **Danish Zargar**, Teaching Assistant, **International Institute of Information Technology**.

Submitted By:

Date: 27/11/2019
Place: HYDERABAD

Vishal Malhotra
Yallamanda Rao Mundru
Anurag Pateriya
Jeevesh Kataria

CERTIFICATE

This is to certify that the project report entitled **"Version Control System"** submitted by **VISHAL MALHOTRA, Yallamanda Rao MUNDRU, ANURAG PATERIYA, JEEVESH KATARIA** has been carried out under my guidance and supervision. The project report is approved for submission requirement for project in 1st semester in **International Institute of Information Technology,HYDERABAD.**

Date: 27/11/2019 **Prof: Shatrunjay Rawat**
Place: HYDERABAD

Version Control System (VJAY)

Project Objective:

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.

Commands Implemented

1. **init**
2. **add**
3. **status**
4. **commit**
5. **rollback**
6. **diff**
7. **log**
8. **retrieve -a vno**
9. **retrieve SHA vno**
10. **Push**
11. **Pull**
12. **Merge**

Version Control system :

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. In case of 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.

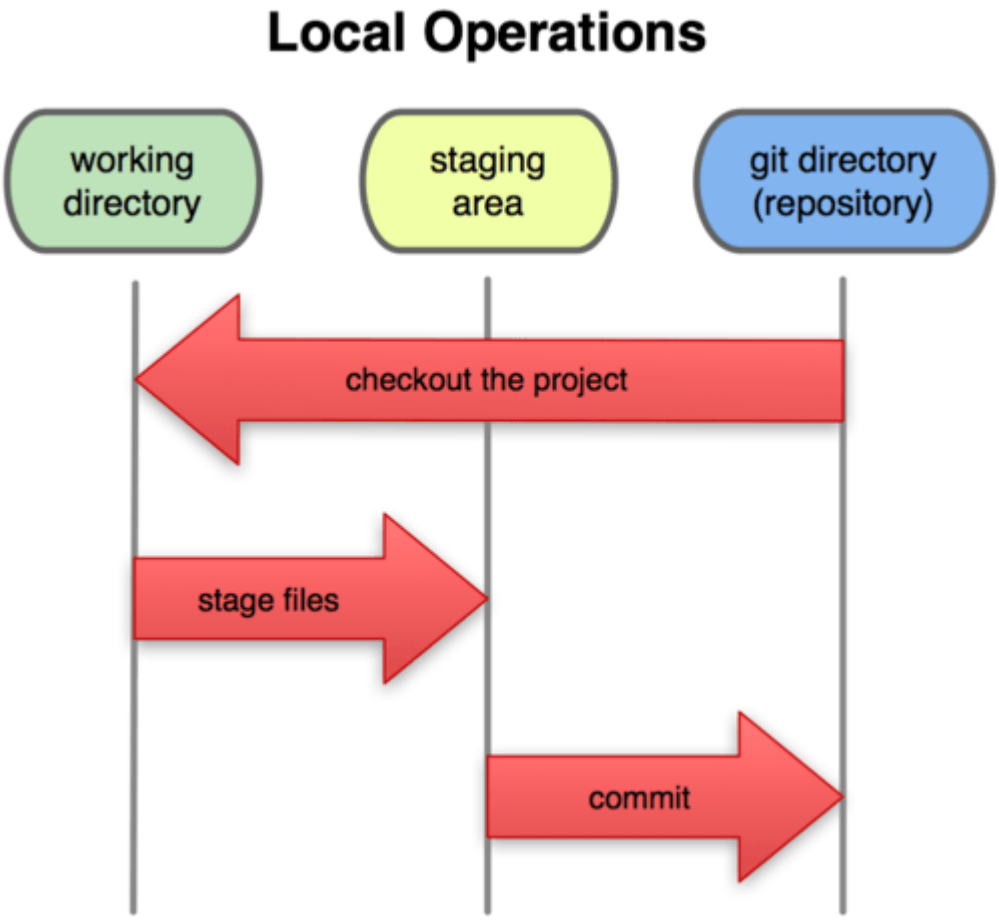
- **Local Version Control Systems :**Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.
- **Centralized Version Control Systems :**The next major issue that people encounter is that they need to collaborate with developers on other systems. These system's have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.This setup offers many advantages, especially over local VCSs. Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.

  The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines.
- **Distributed Version Control Systems :**This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

Git (Version control system):

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

Having a distributed architecture, Git is an example of a DVCS (Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.In addition to being distributed, Git has been designed with performance, security and flexibility in mind.



Commands we have implemented:

- **Init** :

  The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.In this we initialise the directories and files in which we store all the information that helps us in storing metadata regarding various versions like SHA index of files,data belonging to each versions. In order to initialise git we have to run the following command .

  .

```
./a.out init
```

- **Status :**

  The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.In this we compare the SHA of files in present working directory and the SHA of files in previous files.Depending on the result of the above comparison we get 3 set of files

  - Modified files : Files which underwent changes.
  - Untracked files : Files which are newly created
  - Deleted files : Files which are present previously but are deleted.

In order to get the status,we run the following command -

```
./a.out status
```

- **Add :**

  In this we add all the changes to the staging area.Staging area contains the changes made compared to the previous version that are need to considered for next commit.It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run commit.In order to add changes to staging area,we run the following command:

```
./a.out add
```

- **Commit:**

  In this we commit all the changes in the staging area.After this we can consider the data we committed to be permanent in the sense that we can come back to this stage at any time.Content will never be changed unless we do so explicitly.In order to commit the changes,we need to run the following command :

```
./a.out commit
```

- **Rollback :**

  Upon running this, the previous commit which we made will be considered as the latest version. This will come in handy when we made a mistake and we wish to undo all the changes by moving to the previous commit.In order to rollback, we need to run the following command :-

```
./a.out rollback
```

- **Diff:**

  Diffing is a function that takes two input data sets and outputs the changes between them.Git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more. This document will discuss common invocations of git diff and diffing work flow patterns. The git diff command is often used along with git status and git log to analyze the current state of a Git repo.

- **Push**

  The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to

a remote repo. It's the counterpart to git fetch, but whereas fetching imports commits to local branches, pushing exports commits to remote branches. Remote branches are configured using the git remote command. Pushing has the potential to overwrite changes, caution should be taken when pushing. These issues are discussed below.

- **Pull**:

  The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows. The git pull command is actually a combination of two other commands, git fetch followed by git merge. In the first stage of operation 0git pull will execute a git fetch scoped to the local branch that HEAD is pointed at. Once the content is downloaded, git pull will enter a merge workflow. A new merge commit will be-created and HEAD updated to point at the new commit.

- **Merge**

  Merging is Git's way of putting a forked history back together again. The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch

  Git merge will combine multiple sequences of commits into one unified history.In the most frequent use cases, git merge is used to combine two branches. The following examples in this document will focus on this branch merging pattern.In these scenarios,git merge takes two commit pointers, usually the branch tips, and will find a common base commit between them. Once Git finds a common base commit it will create a new "merge commit" that combines the changes of each queued merge commit sequence.