Version Control System

VCS is basically software designed to record changes within one or more files over time. It allows us to undo or to cancel all made or pending changes within one or more files

- It allows maintaining multiple versions of a file, document, program websites etc.
- It keeps track of every modification to the code in a special kind of database.

Why VCS?

- It gives a time machine for going back to the earlier versions.
- Greatly simplifies concurrent works, merging changes.
- Management of changes to files.
- Keep track of what changes occur.
- Allows people to work together.

Kinds of VCS

1. Localized
2. Centralized
3. Distributed

Localized

- Keeps local copies of the files.

Centralized

- There is a server and a client. The server is the master repository which contains all of the versions of the code.
- A central server repository holds the official copy of code, so the connection to the central server is required for most operations.
- svn, cvs, perforce etc

- We make checkouts of the central server to the local copy and make local modifications. The local changes are not versioned.
- Once the local modifications are done, checkin back to the server is done. The check in increments the repo's version.

Drawbacks

- Both the approaches have the drawback, one single point of failure.
- In local VCS, it is the individual computer and in the centralized VCS, it is a server machine. In both systems it is also harder to work in parallel on different features.
-

Distributed version control system

- In a distributed VCS, each user has a complete local copy of a repository on his individual computer.
- We don't checkout from a central repo. We clone it and pull changes from it, like git, mercurial etc.
- The local repo is the complete copy of everything on the remote server.
- Most operations are local,and a central server is not required.
  - Check in/out from the local repo.
  - Commit changes to local repo
  - Local repo keeps version history.

- When everything is done, we can push changes back to the central repository.
- Github is an example of git remote repository.

Ref:
https://serengetitech.com/tech/introduction-to-git-and-types-of-version-control-systems/

What is Git?

- Git is the distributed version control system

- Massively scales, open source
- Most operations are local and it is fast.
- Active community, it has also became the most popular vcs
- It is a Tree history storage system
- It is a content tracking management system
- It provides Ease & Speed.

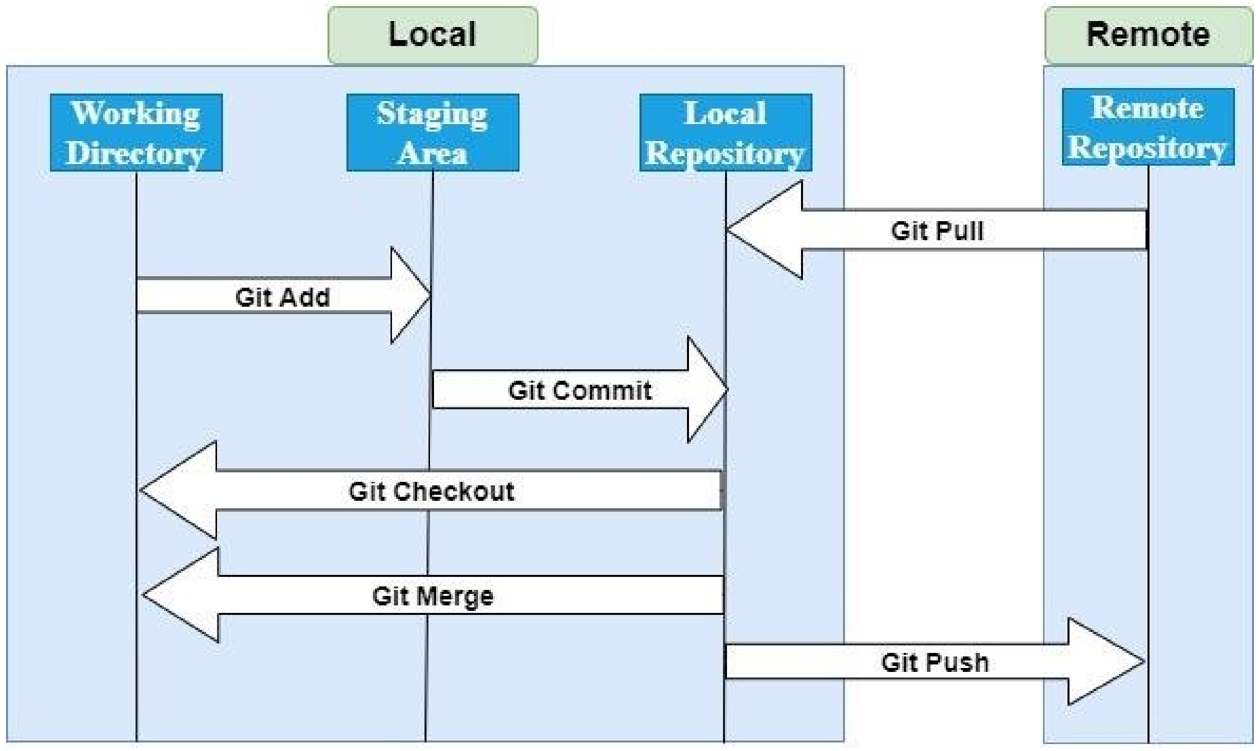
- Ease: Simple to use too and command, also it is a cloud based repository.
- Speed: Support for non-linear development, fully distributed, able to handle large projects.
- Git was created by Linus Torvalds, a creator of Linux in 2005.
- Came out of linux development community
- Designed to do version control on Linux kernel

Key terminologies

- Repository contains files, history, config managed by git.
- Three stages of Git
  - working directory
  - staging area → Also known as pre-commit holding area
  - Commit → Git repository(history)

- Remote repository (Github)
- Master branch → Branches in Git

Basic Git workflow Life Cycle



Installation

Linux
https://git-scm.com/download/linux

Windows
https://git-scm.com/download/windows

Mac
https://git-scm.com/download/mac

$ git version

Git SSH Login
$ cat ./git/config
→ We can see the https url.
The authentication is based on username and password. It is not a good idea, we need to store passwords and chances of exposing the password.

The github provides the ssh based login & almost every remote git repository provides this. The authentication will be based on keys.

To generate ssh keys
$ ssh-keygen

$ cat ~/.ssh/id_rsa_pub
Copy the contents of public keys & provide to github

$ git remote set-url origin git@....
$ git pull

Git Basic Overview

- Starting a project
    - Fresh project(no source yet)
    - Existing source code(locally)
    - Github project(fork & clone)

- Basics overflow(add, commit, push & pull)

- Working with files(rename, move & delete)
- History & Aliases
- Ignoring unwanted files

Starting a fresh project (Local Repository Setup)

Create a project directory

```
$ mkdir demo
$ cd demo
$ git init
$ touch testfiles{1..5}.txt
$ mkdir devops java python
$ touch devops/firstfile.txt
$ git status
$ ls -al
```

→ Creates a .git directory.
→ It contains & maintains all the history, configuration of the git repository.

Note: The empty directory is not tracked by git. Git only keeps track of files, not the directory.

```
$ git status
```

→ Gives the current state of the repository
→ Untracked files will not be considered for committing, we have to make them tracked.

```
$ git add .
```

→. indicates track all files in the current directory. This is called staging.

```
$ git status
```

→ Changes to be committed

```
$ git commit -m "initial commit"
```
→ -m for commit message

Before committing, we need to configure the username and the email address of the user. This is to keep track of who made the changes.
To configure the user and email id.
```
$ git config -- global user.email "emailid@gmail.com"
$ git config -- global user.name "srtimsina"
```

We have global and local configs.
Global → For all the repository
Local → Local to the repository

```
$ git commit -m "initial commit"
```

```
$ git status
```

Adding Git to an existing project (git init)

- Existing source code(locally)

Adding a README.md file

```
$ vim README.md
```
 # Simple Project
First level heading
## Introduction
Second level heading

## Installation
Run the following command to install `net-tools`.

```
apt update
apt-get install net-tools
```
## Purpose

## How to Contribute

Setting up Remote Repository

Create remote repositories on Github, bitbucket, codecommit etc, Github is the most popular remote repository.

Browse https://github.com
If already signed up, login or if new to github signup and create a profile.

Create a repository

Clone the repo to local
```
$ git clone URL
```

Or

Local to remote integration
```
$ cd local_repo
$ git remote add origin
ssh://git@github.com/[username]/[repository-name].git
$ git push
$ git pull (to fetch latest changes)
```

Note: New default branch on Github

- Git still has master as the default branch

- Github has main as the default branch

To reset default branch in Github
Go to setting -> Repositories -> Repository default branch -> Change to desired branch and update

From CLI
```
$ git config --global init.default main
$ git config --global --list
```

To login
https://github.com/login

Create a new repository
→ The repository name should be the same as local so that we can sync both the repos.

Public:
- No authentication required, anyone can access

Private:
- Authentication required, only allowed can access

Create repository
- Push an existing repository from the command line

```
$ cat .git/config
$ cURL
$ git branch -M main
$ git push -u origin main
$ vim demo/testfile1.txt
$ git status
```

$ git add /demo/testfile1.txt
$ git commit -m "add testfile1.txt"
$ git push origin main

$ git log
$ git show (commit ID)
$ git log --oneline
→ To display short message for commit id
$ git show (commit_id)

To delete the repository
First push all changes to remote repo
$ git push --all origin
$ cd ..
$ rm -rf repo_name

Comparison

Comparing working directory & staging area
$ vim testfile1.txt
$ git diff

Comparing working directory & git repository(last commit)
$ git diff HEAD

Comparing between the staging area & the git repository(last commit)
$ git diff --staged HEAD

Limiting comparison to one file(or path)
$ git status
$ git diff
$ git diff – README.md

Comparing between commits

```
$ git log --oneline
 → Shorten the log
$ git diff {commit id} HEAD
$ git diff HEAD HEAD^
HEAD^ is equivalent to HEAD-1
```

Comparing between local & remote main branches

```
$ git diff main origin/main
```

Clean up and push back to Github

```
$ git status
```

Rollback

Method 1

```
$ cd demo
$ ls
$ vim testfile1.txt
→ Add some contents in the existing file
$ cat testfile1.txt
$ git checkout testfile1.txt
$ cat testfile1.txt
```

Method 2

```
$ git status
$ vim testfile1.txt
→ Add some contents
$ git diff
→ Shows the differences
$ git status
```

$ git add .
$ git status
$ git diff --cached
→ Shows the differences in the staged files

To go back or restore from the staged files
$ git restore --staged testfile1.txt

$ git status

To go back or restore from the committed
$ git add .
$ git commit -m "Test commit"
$ git status
$ git diff
$ git diff --cached
$ git log --oneline
$ git diff {prev_commit}..{new_commit}

To get back/ revert to the previous commit
$ git revert HEAD
Or
$ git revert HEAD (previous_commit_id)
→ Correct the revert message if required

$ cat testfile1.txt
$ git log –oneline
→ Gives a new commit id
While reverting it wll still keep the history, However if we need to remove anyways, then we can use.
$ git reset --hard {commit_id_of_old}
$ git log --oneline