Version Control System

Configuration Management ?

- Software Configuration Management(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

- The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

Why do we need Configuration management?

The primary reasons for Implementing Technical Software Configuration Management System are:

- There are multiple people working on software which is continually updating
- It may be a case where multiple version, branches, authors are involved in a software config project, and the team is geographically distributed and works concurrently
- Changes in user requirement, policy, budget, schedule need to be accommodated.
- Software should able to run on various machines and Operating Systems
- Helps to develop coordination among stakeholders
- SCM process is also beneficial to control the costs involved in making changes to a system

Types of VCS

There are Three types of VCS, namely:

1. Local Version Control Systems (LVCS)
2. Centralized version control system (CVCS)
3. Distributed version control system (DVCS)

Local Version Control Systems

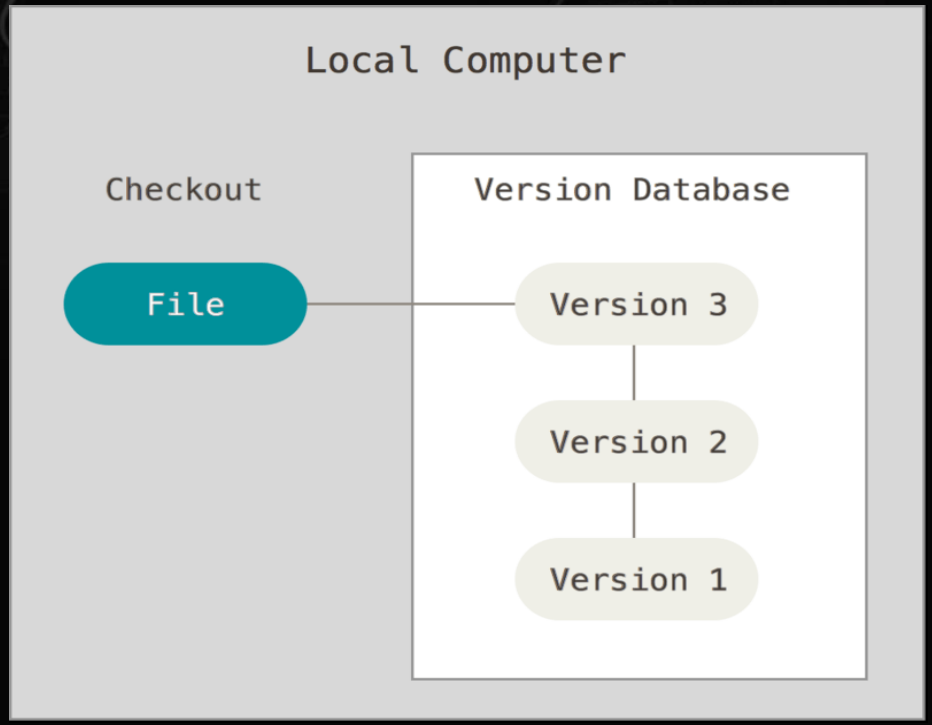
Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever).

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.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

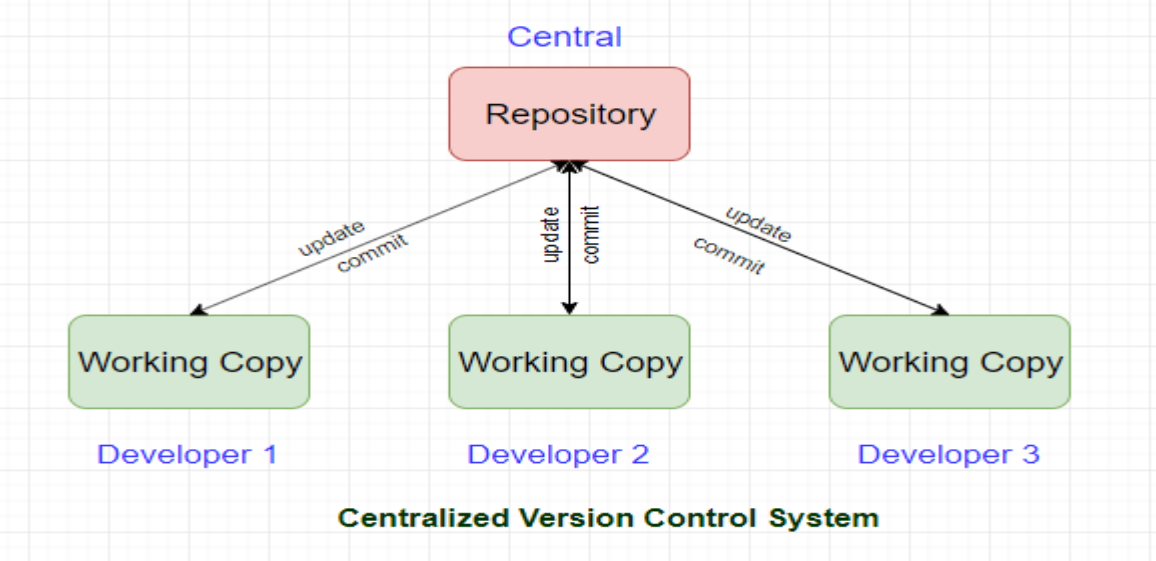
Local Version Control Systems

- One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today. RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.



Centralized Version Control Systems

Centralized version control system works on a Client-Server relationship. Server will have all the information which is transferred to the client. You can say that it is a kid of shared repository which provides latest code to the developers. So developers will be working on master copy every time. It is as simple as you have to pull the latest copy of code, work on it (commit changes) and then push the code back to repository.



Centralized Version Control Systems

The main disadvantage of the Centralized VCS is that it is a single point of failure. If central server went down then you cannot used it. If you are doing remote commit then it will take time as well. You will need internet connectivity to commit any changes.

Examples:

Open source -> Concurrent Versions System (CVS), Subversion (SVN).
Proprietary -> TeamCity, Vault, IBM Configuration Management Version Control (CMVC).

Drawback of CVCS

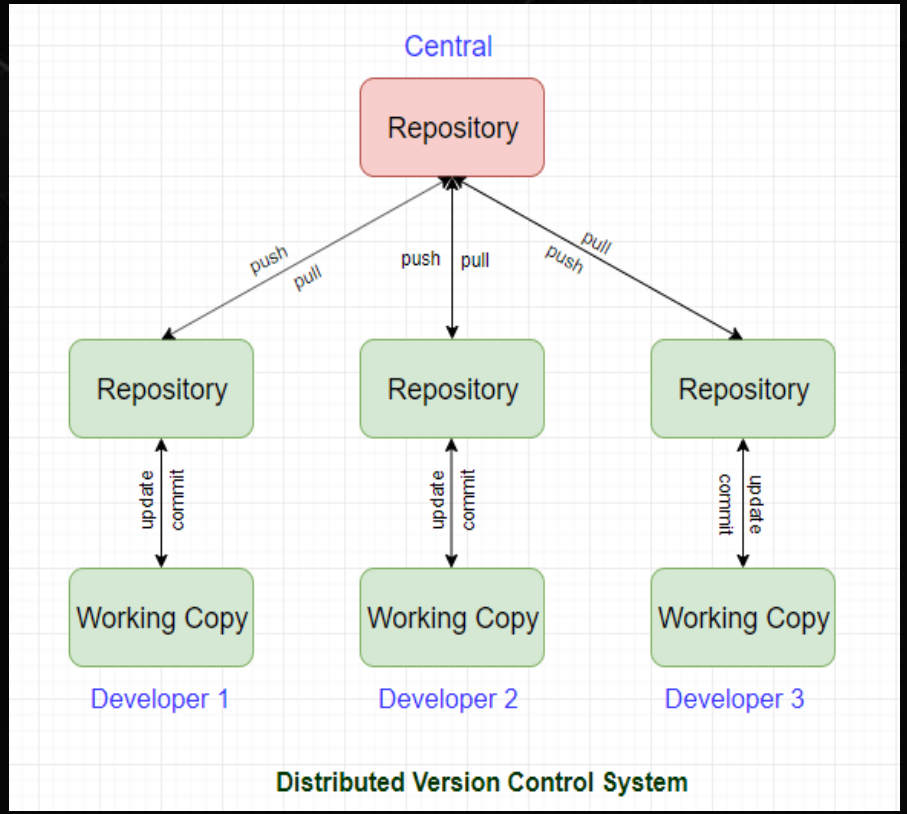
It is not locally available, means we always need to be a connected to network/internet to perform any action.
If everything is centralized, then what happen if central server is failed.

Distributed Version Control Systems

- Distributed version control system has a centralized repository as well as all developers have local copy of repository.
- Developers can work on their local copy simultaneously. They do not required internet connectivity to work on the code. They can do everything in the code except push and pull. If the central server went down then also there will be no impact because of the local repository.

Examples:

- Open source - Git, Mercurial, Bazaar.
- Proprietary - Visual Studio Team Services, Plastic SCM, Bitkeeper



Distributed Version Control System

Difference between CVCS & DVCS

CVCS

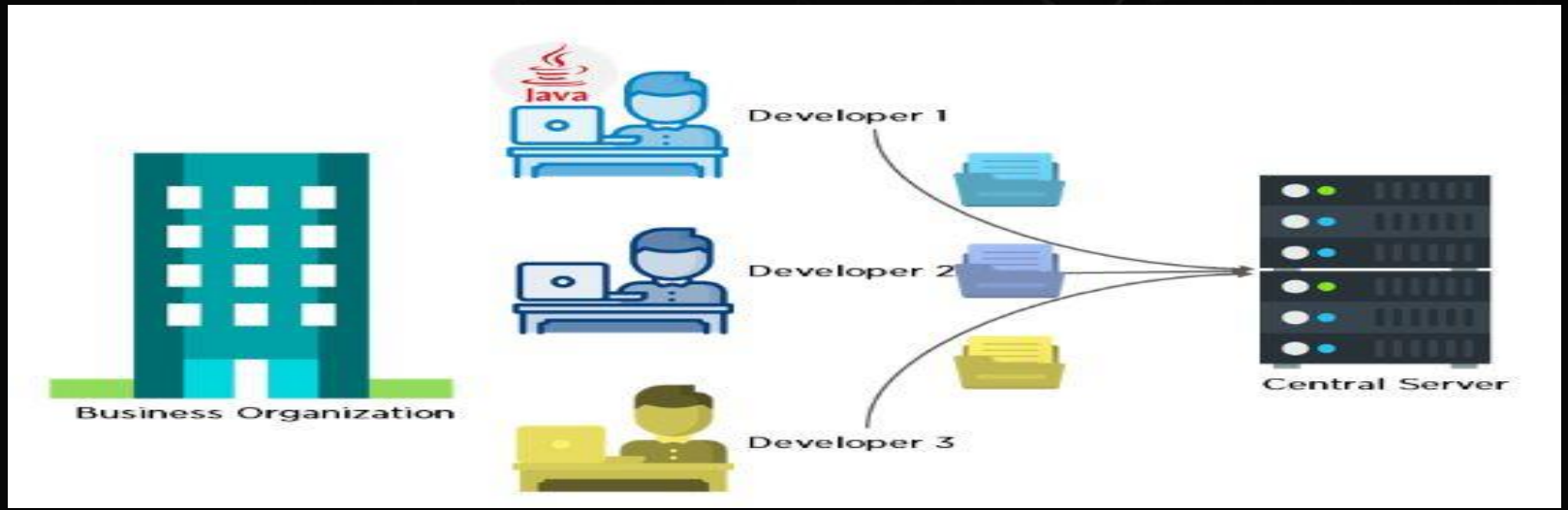
- In CVCS, a client need to get local copy of source from server to do changes.
- CVCS are easy to learn and setup.
- Working on branches is difficult in CVCS developer often face merge conflict.
- CVCS system do not provide offline access.
- CVCS is slower as every command need to communicate with server
- If CVCS server is down, developers can not work.

DVCS

- In DVCS each client can have local data does not required server to do change.
- DVCS system are difficult for beginners to remember multiple commands.
- Working on branches easy to developer less conflict.
- DVCS system allow to work offline cause of local repo.
- DVCS is faster developer work on local copy not required server connectivity.
- DVCS serer is down, developer can work with local copies

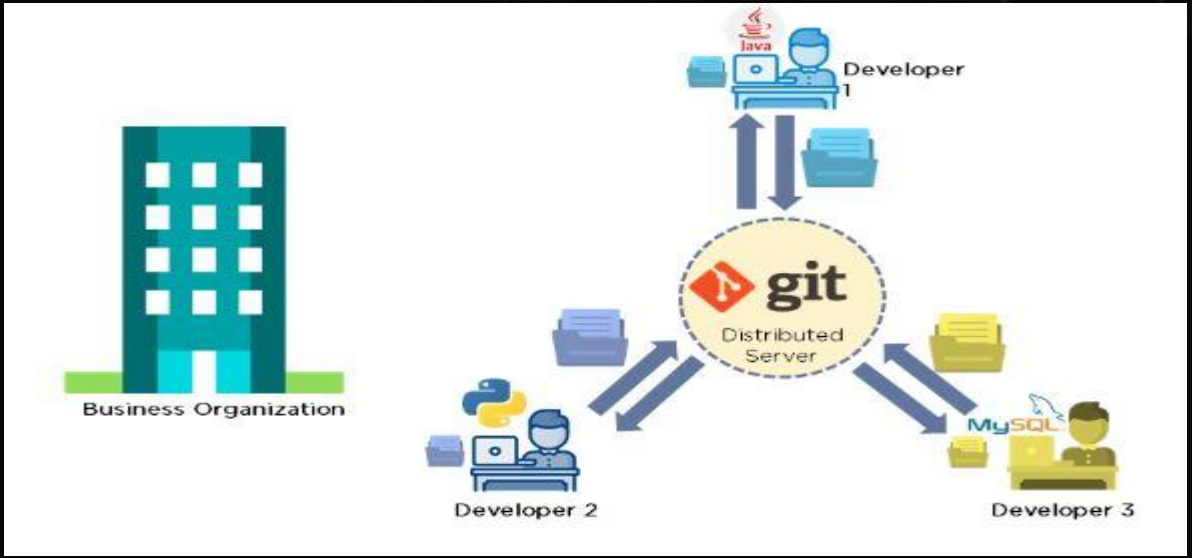
Scenario Before Git

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers



Scenario After Git

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers



A Short History of Git

The Linux kernel is an open source software project of fairly large scope. In linux OS popularly use bitkeeper.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper.
Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development

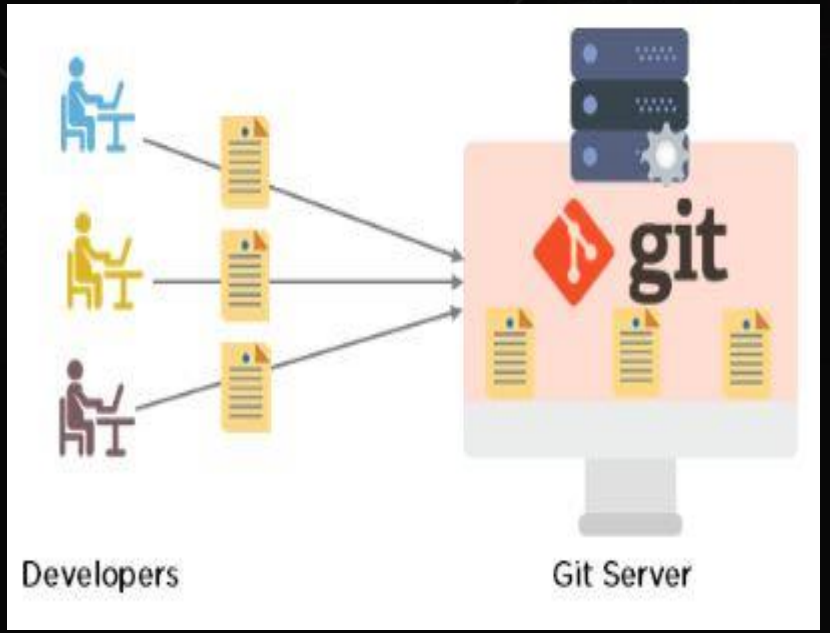
What is Git

Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development.

- Git is used to tracking changes in the source code
- The distributed version control tool is used for source code management
- It allows multiple developers to work together
- It supports non-linear development through its thousands of parallel branches

Feature of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Supports collaboration
- Branching is easier
- Distributed development



Stages of git (workflow) and its terminology

After install git,initialise it (git init - startup git software),it convert directory(MyMIgit) into Repository and it devided into three logical areas (called local repository)

- Working/workspace Direcoty
- Staging Area
- Local Repo.

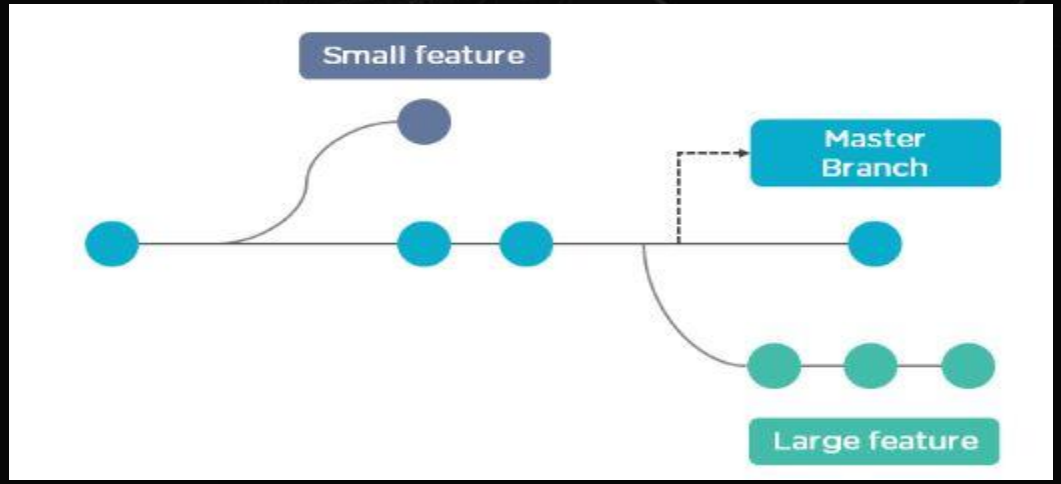
Every time when any one update in code new commit ID will be generate, which is very useful to check who,when,what updated in code or also use to access old code with help of commit ID.

Note:

Snapshot take here only the incremental backup means commit take only incremental backup

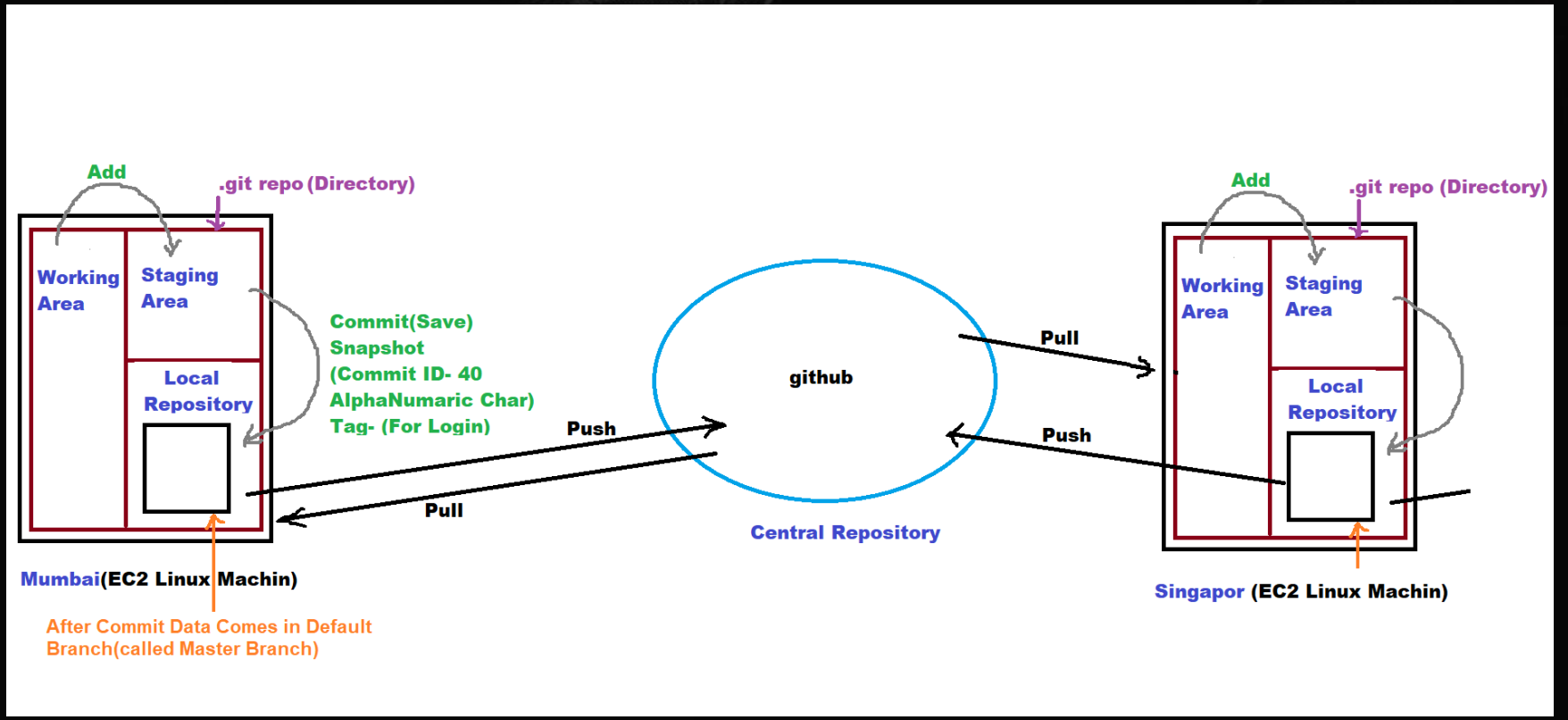
Branch of Git

Branch in Git is used to keep your changes until they are ready. You can do your work on a branch while the main branch (master) remains stable. After you are done with your work, you can merge it with the main office.



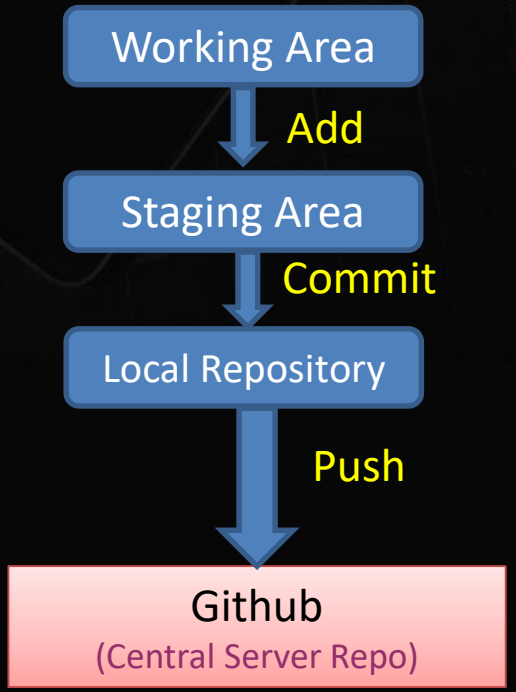
The above diagram shows there is a master branch. There are two separate branches called “small feature” and “large feature.” Once you are finished working with the two separate branches, you can merge them and create a master branch.

Stages of git and its terminology



Stages of git and its terminology

In other CVCS, developers generally make modifications and commit their changes directory to the Repository. But bit uses track each and every modified file. Whenever you do commit an operation, git look for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files



Stages of git and its terminology

Repository:

- Repositoy is a place where you have all your codes or kind of folder on server.
- Git is a kind of folder related to one product.
- Changes are personal to that product repository.

Server:

- It store all repositories.
- It contains metadata also

Working Directory:

- Where you see files physically and do modification.
- At a time, you can work on particular branch.

Stages of git and its terminology

Commit:

- Store Changes in repository. You will get one commit-ID
- It is 40 alpha-numeric Characters.
- It uses SHA-1 checksum concept.(Integrity)
- Even if you change one dot(.), commit-ID will get change.
- It actually helps you to track the changes.
- Commit is also named as SHA1 hash.

Commit-ID/Version-ID :

- Reference to identity each change.
- To identify who changed the file.

Stages of git and its terminology

Tags :

– Tags assign a meaningful name with a specific version in the repository. Once a Tag is created for a particular save, even if you create a new commit, it will not be updated.

Snapshot :

– Represents some data of particular time.
– Git is always incremental backup. i.e. it store the changes (append data) only. Not entire copy of data

Push :

– Push operations copies changes from a local repository instances to a remote or central repo. This is used to store the changes permanently into the git repository.

Stages of git and its terminology

Pull :

- Push operations copies the changes from a remote repository to local repository (local machine).
- Pull operations is used to synchronization between two repository (local or remote repo)

Branch:

- Product is some, so one repository but different task.
- Each task has one separate Branch.
- Finally merges (code) all Branches.
- Useful when you want to work parallel.
- Can create one branch one the basis of another branch.
- Changes are personal to that particular branch.

Stages of git and its terminology

- Default branch is called Master.
- File created in workspace will be visible in any of the branch.
- Workspace until you commit once you commit, then that file belongs to that particular branch

Type of repository

Bare Repositories (Central Repo)

- It store and share only
- All central repositories are bare repo

Non-Bare Repositories (Local Repo)

- Where you can modify the files all local repositories are non-bare repositories.

Advantages of git

- Free and open source.
- Fast and small as a most of the operations are performed locally, therefore it is fast
- Security - Git uses a common cryptographic hash function called secure hash function (SHA-1) to name and identify objects within its database.
- No need of powerful hardware.
- Easier branching – if we create a new branch, it will copy all the codes to the new branch.

Git commands

- Create Repositories
  git init
- Make Changes
  add
  commit
  status
- Parallel Development
  branch
  merge
  rebase
- Sync Repositories
  push
  pull
  add origin