




What is Version Control System?



Version control system

- It is a software which maintain the record of every change and help to create , built , delete and edit the code.
- **Feature's:-**
 - **Transparency-** It can check which whom when can make the changes in code .
 - **Security and backup-** It has a different copy of it either on local server or remote server.
 - **Version control-** It kept the record of every version that has changed or present version of any code.
 - **Collaboration-** Multiple developers can work simultaneously on a same project .

Why Version Control System?

- When Version control was not available all the developer face multiple problem like, there storages get full due to multiple updated version of code , Developer has to share their code manually which leads time delay in project and sometime they can't find the error due to manually exchange of code there is no transparency what when is done by whom data make get lost if the system get corrupt.
- Then version control come in picture and it solve the problem which we facing earlier.

Types of version control system

- There are four types of version control system
 - a.) Manual
 - b.) Local
 - c.) Centralized
 - d.) Distributed

a.) **Manual**- Developer can copy the files anywhere in the server and forget the directory or folder so it overwrite the file.

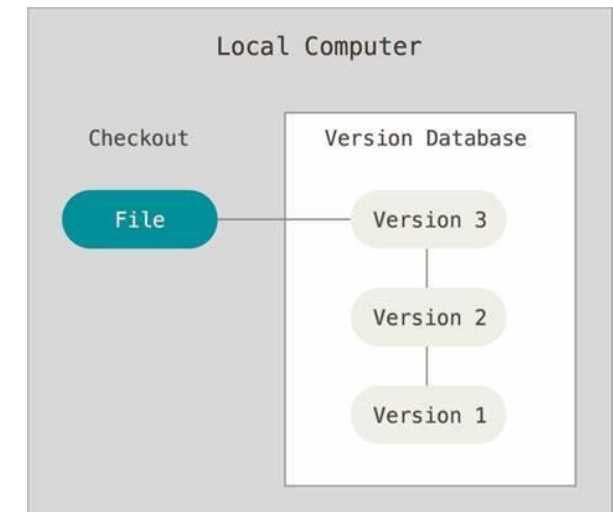
Disadvantage-

- Cant have the idea of location
- Hard to follow the path


b.) **Local**- It has the track of every file in system with their specified version's.

Disadvantage-

- Not ideal for collaboration.



c.) **Centralized**- An organization/platform has multiple files in it which can be accessed centrally. If any developer need it , it can access from platform then commit merge or push the changes after that any developer can work accordingly.

- 
- Developer workstations connected to the server to retrieve files for editing are called clients.

Advantages-

- It has the history of every change that happen to code and every user can see it from where they are.
- It is better than local it can be managed easily and restrict what not to do.

Disadvantage-

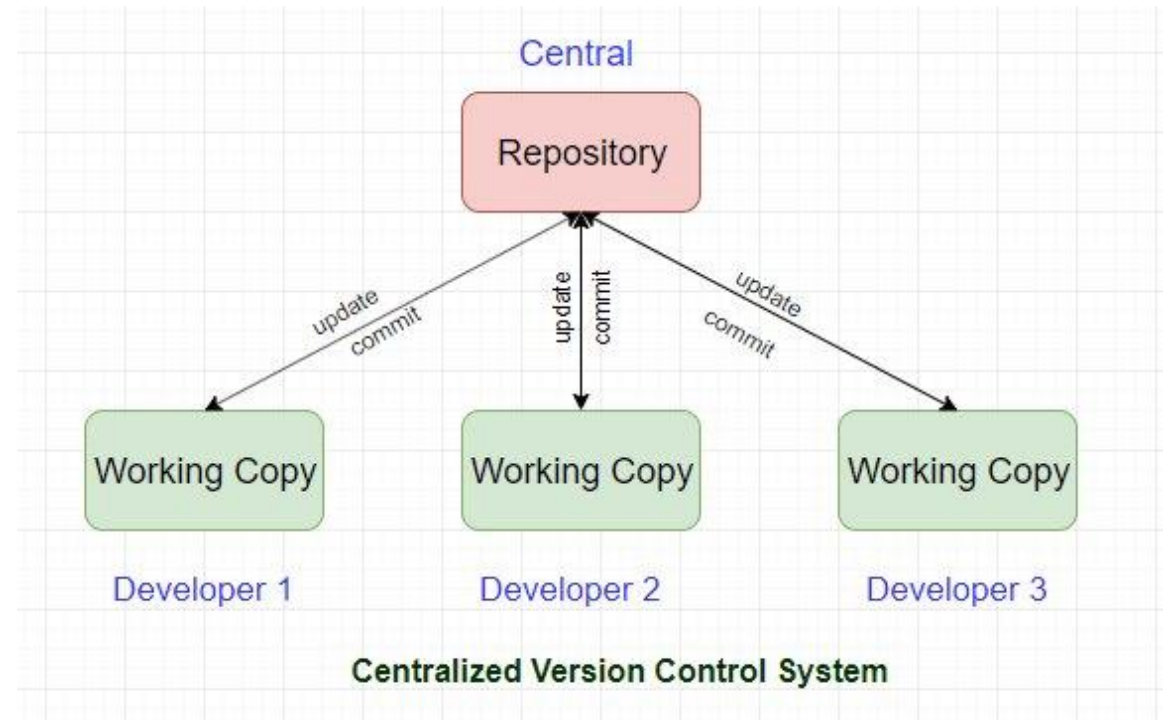
The data is saved only on central remote server if it get lost or corrupt then data is lost as it doesn't have any other copy it is wastage of every developer work.

No one can work together.

It required always internet to work and it is slower.

How data get lost in CVCS?

- Data get lost in CVCS because of the remote server if the original get corrupted or server get down for time without saving the chnges it get lost so its unsafe to use it.



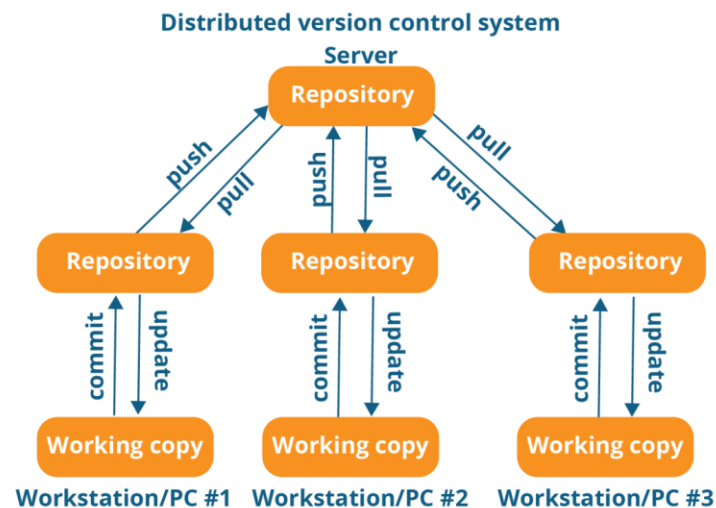
d.) **Distributed**- Each developer working on the project has a complete copy (clone) of all files and their entire change history on their local machine.

Advantage -

It is faster and doesn't required internet access to complete the work.

Multiple developer can work together .

Every developer has a copy of there work in their local as well.



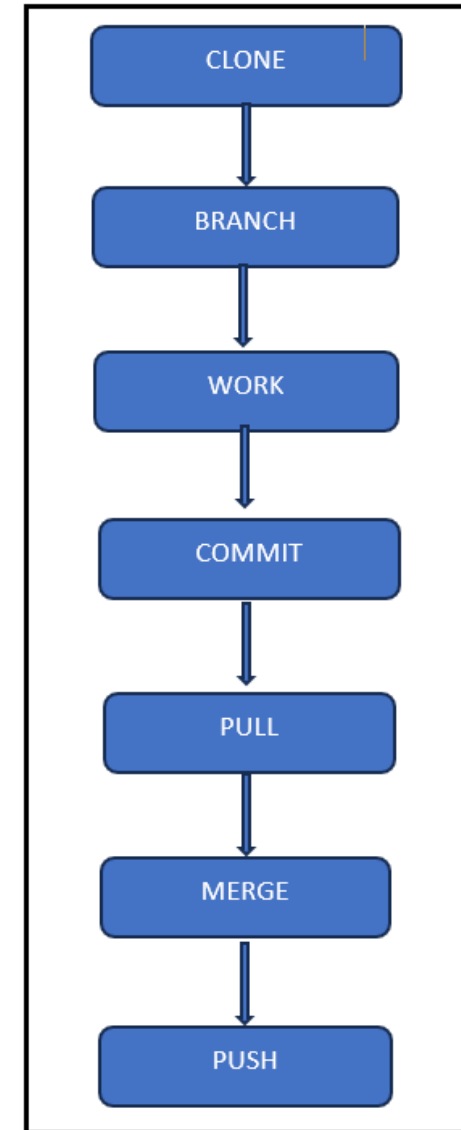
Git Version Control

It is a distributed version control system and it is a software which maintain the record of every change and help to create , built , delete and edit the code.

Features-

- Version control
- Merging
- Branching
- Remote repositories
- Distributed

Git Workflow



Git Installation

- For windows we download git bash in our local server to run the local git command.
- Downloading vs code for the code editor.
- After installing git bash first check the version of git in git bash so that you are sure that git has been downloaded using -

Git -version or git -v

- Create a git folder in local or in vs code to start the process.
- First list all the git setup commands after that start the git workflow.

Git Setup

a.) Setting Up Your Identity- Git tracks changes by the identity of username or email address so it is necessary to configure yourself in git to track and commit changes. We use simple syntax in this:-

1. `git config --global user.name "name"`
2. `git config --global user.email "emailid"`

These commands make the changes globally if we don't want to use it globally then we can it as locally just by removing global from syntax.

b.)Configure The Default Editor- When Git requires you to enter a message for a commit or a merge, it opens a text editor. You can set your preferred editor using:

```
git config --global code.editor "code-wait" {code  
wait is used for vs code}
```



c.) Git Color Setup- We can use different color in git for their branch , commit and status.

```
git config --global color.ui true
```

```
git config --global color.branch auto
```

```
git config --global color.status auto
```

d.) List Git Setting- To list all the changes we have made so far use the simple git command.

```
git config --list
```

Stages of Git

- There are three stages in git -

a.) Working Directory – Where the code is written by developer and clone from the remote repository.

Git clone "http link"

b.) Staging Area – Also known as Index. Here all the changes get add if we a change in any file present in working directory then it get add and it asked for committing those changes.

Git add filename1 (for one file)

Git add filename1 filename2 ..(for multiple files)

Git add . (if multiple files are present in same place)

c.) Local Repository- It is a folder present in the local system where all the code get clone or written by developer and if changes occur then they get added after committing those changes your local repository is update to date .

Types of File-

- There are four types of file which get formed while we create , delete, replace anything
 - a.) Untracked – It mean they are not getting tracked through git.
 - b.) Modified- These files formed during add or commit anything in repository.
 - c.)Unmodified – These files are unchanged it occur after we add , commit the repo and again we check status the files are unmodified nothing to commit.
 - d.)Staged- The file is ready to commit.

Git commit -m "message"

- Here -m stand for modified which mean this file is modified and we made some changes in it what are those changes can be remember through message.

Workflow From Remote To Local Repository-

- **Steps-**

Github repository

Clone

Code

Status

Add

Commit

Branch

Merge

Status

Add, commit

Push

Workflow From Local to Remote Repo

- Steps-

Folder creation

Git init

Creating a file and a code

Git status

Add , commit

Git remote add origin http link

Git remote -v

Git branch

Git branch -M main

Git status

Git push -u origin main

(if some changes are made on remote and we want to pull those changes in local then we use)

Git pull -u origin main

Commands-

- **Git branch-** used to create a branch for that particular work

Git branch (check the number of branches)

Git branch -M branch name (rename the branch name)

Git checkout -b branch name (create a new branch)

Git checkout (switch to that particular branch)

Git branch -d branch name (delete that particular branch)

Git branch -merged (showed all the merged branches)

- **Git difference** – show diff btw two branches

Git diff branch name (show the diff btw two branches)

Git diff -staged (it show the diff after staging mean adding before committing)

- **Git push** – used to push the changes made in remote repo or used to push the local repo to remote repo.

Git push -u origin main (-u is upstream if we defined this then we don't have to use the origin main again and again, origin is the name of repository and main is the branch name to which we are pushing the code we can change it if we have another branch to push)

- **git pull** – It download and copy all the changes from remote. It used to pull the repo from remote to local if we made some commit in remote and want to update our local repo commit .

Git pull -u origin main

- **Git fetch** – It is used to tell what changes made and when , by whom without merging them.

Git fetch (will tell about all the branches)

Git fetch -u remote branch (which branch and origin changes you want to see)

~ If we create a branch in remote and get the information from git fetch then if we do git checkout -b branch name the same branch will show in our repository.

- **Git merge** – It is used to integrate changes btw two branches . After developing a feature or fixing a bug on a separate branch, we typically merge that branch back into the main branch to include those changes in the main codebase. When we merge branches it create extra commit.

Git merge branch name(the branch we want to merge)

- **Git Rebase**- It is also used to merge the branches but we cant merge the branches directly we applying commits from one branch right on top of another to create a linear history, usually without generating a merge commit.

Git rebase branch name

git rebase -i HEAD~3

- **Git revert** – It create a new commit that undoes the changes from a specified previous commit without changing the history commit.

Git revert commit-id

- **Git log** – it is used to show the history of git which shows the commit id changes , author date

Git log

Git log --oneline(will show all the details in one line)

- **Git stash**- it is used to to save the unstaged work if we want to switch from one branch to another branch .

Git stash

Git stash pop (pop is used to undo all the changes after switching to that particular branch)

- **Git remote** – it is used to show all the details of remote repo to local without accessing it.

Git remote show origin (show all branches and commit in origin)

- **Git cherry-pick** - to choose and apply specific commits from one branch to another. It is used for bug fixing commit if we fix a bug and it is available in all three branches then then we pick commit id and apply that commit in all the branches instead of merging them .

It is used when we accidentally made some commit in wrong branch.

Git cherry-pick hash-id

Git cherry-pick –abort (to abort the commit)

Git cherry-pick –continue (if some merge conflicts appear and after solving them we used continue to continue with the commit)

Git cherry-pick – quiet (want to stop the present commit)

Git cherry-pick hash-id –e (want to edit the commit id msg)

Merge Conflicts-

If the same part of the same file was edited differently by both master branch and the feature branch. It show error which is called merge conflicts and it get solved by two ways-

- a.) Manually (in vs code we remove extra character manually)
- b.) Mergetool (to start a visual merge tool that helps with conflict resolution)

Git mergetool