Git and Git hub

Centralized Version Control System (CVCS):

- Here, a client needs to get local copy of source from server, do the changes and commit those changes to central source on server.
- CVCS system are easy to learn and setup.
- Working on branches is different in CVCS. Developer often faces 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 cannot work.

Distributed Version Control System (DVCS):

- In DVCS, each client can have a local branch as well and have a complete history on it. Client need to push the changes to branch which will then be pushed to server repository.
- DVCS systems are difficult for beginners. Multiple commands need to be remembered.
- Working on branches is easier in DVCS. Developer faces less conflict.
- DVCS system are working fine on offline mode as a client copies the entire repository on their local machine.
- DVCS is faster as mostly user deals with local copy without hitting server every time.
- If DVCS server is down, developer can work using their local copies.

Repository:

- Repository is a place where you have all your code or kind of folder on server.
- It is kind of folder related to one product.
- changes are personal to that particular repository.

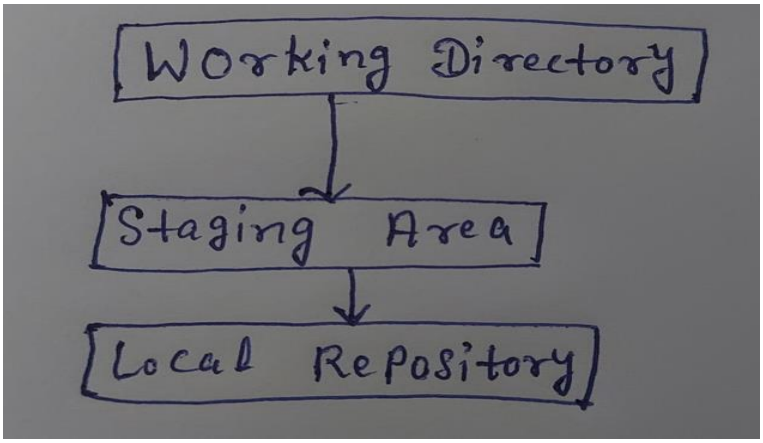
Server:

- It sores all repositories.
- It also contains metadata.

Working Directory:

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

- In other CVCS, developers generally make modifications and commit their changes directly to the repository.
- But git uses a different strategy. Git does not track each and every modified file. Whenever you commit any operation, git looks for the file present in the staging area.
- Only those files present in the staging area are considered for commit and not all modified files.



Commit-ID/Version-ID/Version:

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

Tags:

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

Snapshots:

- Represents some data of particular time.
- It is always incremental i.e It stores the changes (append data) only, not entire copy.

Commit:

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

Push:

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

Pull:

- Pull operation copies the changes from a remote or central repository to a local machine.
- The pull operation is used for synchronization between two repositories.

Branch:

- Product is same, so one repository but different task.
- Each task has one separate branch.
- Finally merges (Code) all branches.
- Useful when you want to work parallelly.
- Can create one branch on the basis of another branch.
- Changes are personal to that particular branch.
- Default branch is “Master”.
- File created in workspace will be visible in any of the branch work space until you commit.
- Once you commit, then that file belongs to that particular branch.

Advantages of Git:

1. Free & open source.
2. Fast & small -> As most of the operations are performed locally, therefore it is fast.
3. Security -> It uses a common cryptographic hash function called Secure Hash Function (SHA1) to name & identify objects within its database.
4. No need of powerful hardware.
5. Easier Branching -> If we create a new branch, it will copy all the codes to the new branch.

Types of Repositories:

Bare Repositories (Central Repo):

- Store & share only.
- All central repositories are Bare Repo.
- Ex- Git Hub, Git Lab, Bitbucket etc

Non-Bare Repositories (Local Repo):

- Where you can modify the files.
- All local repositories are Non-Bare Repositories.

Commands for Git:

At local machine:

- sudo su (Switch to root user)
- yum update -y (Update the packages)
- yum install git -y (Install git)
- git --version (Check git version)
- git config --global user.name “abhay” (Specify user name)
- git config --global user.email “abhay@gmail.com” (Specify user email)
- git config --list (To verify the username & email)

Note: User name & email address will be reflected if we make any modification & push it to central repo for identification purpose. If there is any mistake with code, other person can send email to my email id.

On 1st Machine:

- Create one directory & go inside it.
- git init
- touch myfile (Put some data)
- git status
- git add <file>
- git status
- git commit -m “1st commit from machine 1”
- git status
- git log
- git show <commit-id>
- git remote add origin <central git url>
- git push -u origin master (‘-u’ to specify username & then push the code)
- git push origin master (Alternate way to push)

Note: Enter Git hub (central repo) username & password.

On 2nd Machine:

- Create one directory & go inside it
- git init
- git remote add origin <git hub repo url>
- git pull origin master (Pull the code)
- git log

- git show <commit-id
- git status
- git add .
- git commit -m "1st commit from machine 2"
- git status
- git log
- git log -1 [To see latest or last commit only]
- git log -2 [To see last two commits only]
- git log --oneline [To see all commits in one line]
- git push origin master (Push the code)

To Ignore Some Files While Committing:

Create a hidden file .gitignore and enter file format which you want to ignore.

For eg->

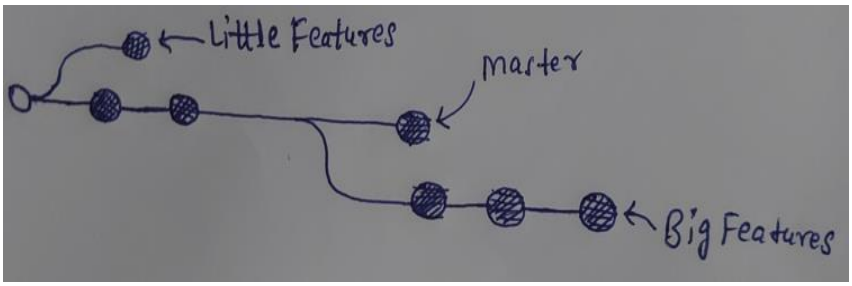
- vim .gitignore
  - *.css
  - *.java
- git add .gitignore
- git commit -m "Latest update exclude css."
- git status

Create some text, java & css files and add them by running "git add".

For eg->

- touch file1.txt file2.txt file3.java file4.css
- ls
- git status
- git add .
- git status
- git commit -m "My text files only."

Branch:



The diagram above visualizes a Repository with two isolated lines of development.

One for a little features & one for a longer running features. By developing them in branches, it's not only possible to work on both of them in parallel, but it also keeps the main Master Branch free from error.

- Each task has one separate branch.
- After done with code, Merge other branches with Master.
- The concept is useful for parallel development.
- You can create any no. of branches.
- Changes are personal to that particular branch.
- Default branch is 'Master'.
- Files created in workspace will be visible in any of the branch workspace until you commit. Once you commit, then that file/files belongs to that particular branch.
- When created new branch, data of existing branch is copied to new branch.

Git Branch Commands:

- git log --oneline
- git branch [To see list of available branches]
- git branch <branch name> [To create a new branch]
- git checkout <branch name> [To switch branch]
- git branch -d <branch name> [To delete a branch]

Merge:

- You can't merge branches of different repositories.
- We use pulling mechanism to merge branches.

Git Merge Commands:

- git merge <branch-name> [Always do merge from Master branch]
- git log [To verify the merge]
- git push origin master [To push to central repository like git hub, git lab etc]

Git Conflict:

- When same file name having different content in different branches, if you do merge, Conflict occurs.
- Conflict occurs when you merge two branches.
- So, resolve conflict then add & commit.

Git Stashing:

- git stash [To stash an item]
- git stash list [To see stashed items list]
- git stash apply stash@{n} [To apply nth stashed item. Zero will be the latest]

Then you can add & commit.

- git stash clear [To clear the stash items]

Git Reset:

It is a powerful command that is used to undo local changes to the state of a git repo.

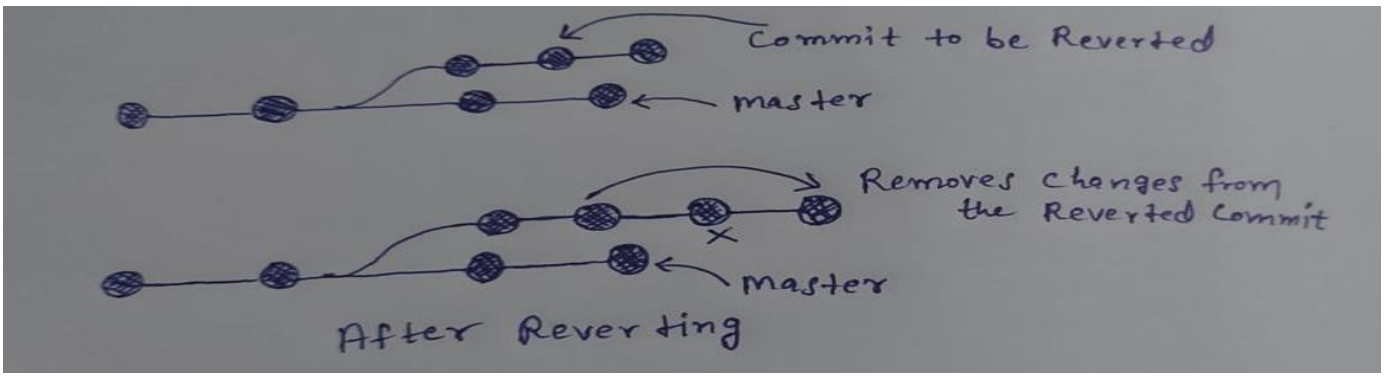
- git reset <file-name> [To reset staging area]
- git reset . [To reset staging area]

To reset the changes from both staging area & working directory at a time-

- git reset --hard [It will delete the file as well]

Git Revert:

- The revert command helps you undo an existing commit.
- It does not delete any data in the process.
- Instead, rather git creates a new commit with the included files reverted to their previous state. So, your version control history moves forward while the state of your file moves backward.
- Reset -> Before commit
- Revert -> After commit



Git Revert Commands:

- git status
- cat > newfile

Hi, final code for app

- git add .
- git commit -m “code”
- git log --oneline
- git revert <commit-id> [Put comment here inside this file as, (Please ignore previous commit)]

How to Remove Un-tracked files:

- git clean -n [Dry run: It will only give you warning message, but won’t remove]
- git clean -f [Forcefully Remove]

Tags & Commands:

Tag operation allows giving meaningful name to a specific version in the repository.

- git tag -a <tag-name> -m <”message”> <commit-id> [To apply Tag]
- git tag [To see the list of tags]
- git show <tag-name> [To see particular commit content using tag]
- git tag -d <tag-name> [To delete a tag]

Git hub Clone:

- Open git hub website.
- Login & choose existing repository.
- Now, go to your Linux machine & run below command-

  Git clone <URL of git hub repo>

- It creates a local repo automatically in Linux machine with the same name as in git hub account.

Note: We can do all these things in Git Hub Website if we wish to do it graphically.