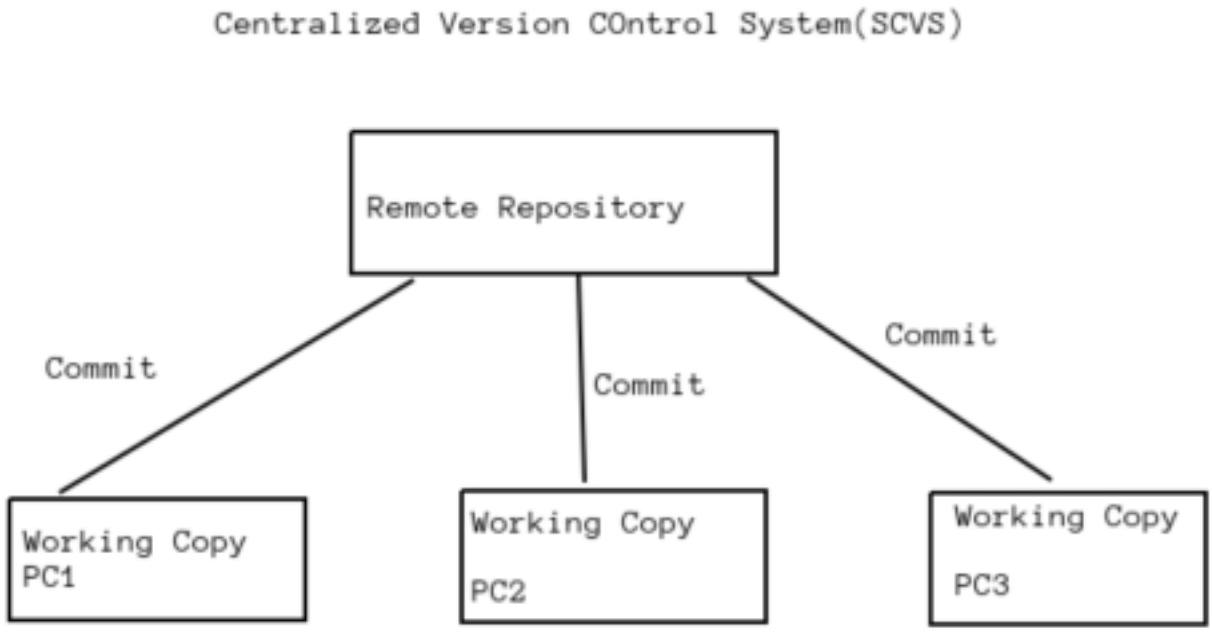
# *3.Git*

## Introduction to GIT :

• **Git** is a version control system that lets us manage and keep track of our source code history
• Git is created by Linux Torvalds
• Github & Gitlab are cloud-based hosting services that lets us manage **Git** repositories.

*Source Code Management (OR) Software Configuration Management :*

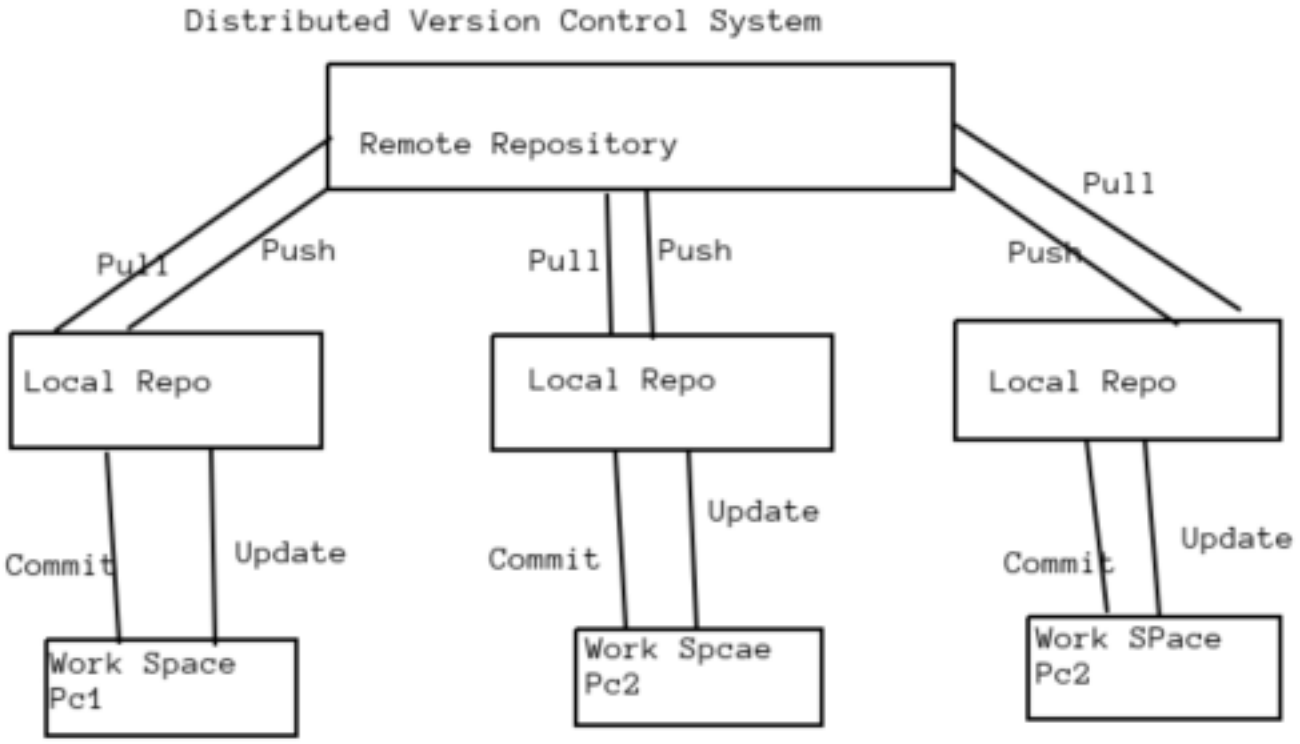
• Centralized Version Control System : Eg : SVN Tool




Drawbacks :

□ It is not locally available,meaning we always need to be connected to a network to perform any action
□ As everthing is centralized,if central server gets failed,we will loose entire data

• Distributed Version Control System :




□ In distributed version control system,every contributor has a local clone of main repository i.e everyone maintains a repository of their own which contains all the files & metadata present in main repository

• CVCS VS DVCS

| CVCS | DVCS |
|---|---|
| Client need to get local copy of source from the server,do the changes and commit those changes to the central source on the server | Each client c push the cha |
| These are easy to learn and setup | Difficult for b |
| Working on branches are difficult.Developers often face merge conflicts | Working on b |
| These do not provide offline access | DVCS systen the local ma |
| CVCS is slow because every command needs to be communicated with central server | DVCS is faste |
| If CVCS server is down,developer cannot work | If DVCS serv |

## Stages of Git & Its Terminology

*Repository :*

• Repository is a place where we have all our codes or kind of folder on server
• It is a kind of folder related to one product
• Changes are personal to that particular repository

### Server :

• It stores all repositories
• It contains metadata als

### Working Directory :

• Where we see files physically and the modification
• At a time,we can work on particular branch
• In other CVCS,developers generally makes 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 we do commit an operation git looks 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

Working Directory → Staging Area → Local Repository → Github

### Commit :

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

### Commit-ID/Version-ID/Version :

• Reference to identify each change
• TO identify who changed the file

### Tags :

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

### Snapshots :

• Represents same data of particular time
• It is always incremental i.e It stores the changes (appended data) only,not entire copy

### 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

Pull :

• Pull operation copies the changes from the remote repository to a local machine.The pull opeartion is used for synchronisation between two repos

Branch :

• Product is same,so one repository but different task
• Each task has one separate branch
• Finally merges (code) all branches
• Useful when we 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 workspace untill we commit.Once commit,then that file belongs to that particular branch

**Advantages of Git :**

• Free and Open Source
• Fast and small as 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 databases
• No need of powerful hardware
• 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 and share only
□ All central repos are BAre repos

• *Nonbare repositories(Local Repo)*
□ Where we can modify the files
□ All local repos are non-bare repos

*Create Linux mahines one in Mumbai and another one in Singapore region using AWS EC2 Instances*

**Install Git in Linux Machine**

• Command to Use :(Do in both Machines)
□ sudo su
□ yum update -y
□ yum install git -y
□ git --version
□ git config --global user.name "Username"

□ git config --global user.email “example@gmail.com”

• *Create and Verify Github Account*

**Commit,Push & Pull from Github :**

• *Commands to use in* Mumbai Region :
□ Login into Mumbai EC2 Instance
□ Create one directory and go inside it → mkdir mumbaigit
□ git init
□ touch myfile → add some data
□ git status
□ git add .
□ git commit -m “1st commit from Mumbai”
□ git status
□ git log
□ git show <commit-id>
□ git remote add origin <central git url>
□ git push -u origin master(enter username & password)
□ cat >>myfile → modify content
□ git status
□ git add .
□ git commit -m “2nd commit from Mumbai”
□ git status
□ git log
□ git show <commit-id>
□ git push origin master(enter username & password)

• Commands to Use in Singapore EC2 Instance :
□ Create one directory and go inside it → mkdir singaporegit
□ git init
□ git remote add origin <central git url>
□ git pull -u origin master
□ git log
□ git show <commit-id>
□ cat >>myfile → add some content
□ git status
□ git add .
□ git status
□ git commit -m “Singapore update 1”
□ git status
□ git log
□ git show <commit -id>
□ git push origin master(enter username & password)

• *Commands to use after Above steps in* Mumbai Instance :
□ cd mumbaigit
□ git pull origin master

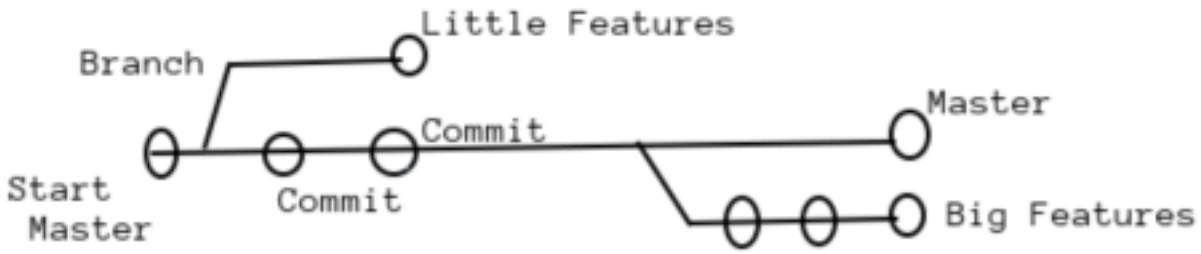
□ git log
□ git show <commit id>

Gitignore : .gitignore

• To ignore some files while committing
• Create one hiddenfile .gitignore and enter file format which we want to ignore
• Ex : vi .gitignore → enter *.css *.java

• Commands to create .gitignore and add to Centralrepo :
□ cd mumbaigit
□ vi .gitignore → enter *.css *.java
□ git add .gitignore
□ git commit -m “added .gitignore”
□ git status
□ touch file1.text file2.txt file3.java file4.css file5.java
□ git status
□ git add .
□ git status
□ git commit -m “Latest updated excluded css & java files”
□ git log
□ git show <commit id>
□ touch file6.java
□ git status
□ touch file7.txt
□ git status
□ git log -1
□ git log -2
□ git log --oneline
□ git log --grep “ignore”
□ git log --grep “commit”

**Git Branches :**




• The diagram above visualizes a repository with two isolated lines of development.
• One for a little features,and one for a longer-running features.
• By developing them with Branches,its not only possible to work on both of them in parallel,but it also keeps the main branch free from error

• Each task has one separate branch
• After done with code,merge other branches with master
• This concept is useful for parallel development
• We can create any no of branches
• Changes are personal to that Particular Branch
• Default branch is “Master”
• File created in workspace will be visible in any of the branch workspace untill we commit.Once we commmit,then that files belongs to that particular branch
• When created new branch,data of existing branch is copied to new branch.

*Important Commands of Git Branches :*

□ git branch → To see list of available branches
□ git branch <new branch name> → To Create a New Branch
□ git checkout <branch name> → To switch to another branch
□ git branch -d <branch name> → To delete branch

*Commands to be used in Lab :*

□ sudo su
□ cd singaporegit
□ git log --oneline
□ git branch
□ git branch branch1
□ git  branch
□ ls
□ git checkout branch1
□ git branch

□ ls
□ git log --oneline
□ cat >branch1file → add content and save
□ ls
□ git add .
□ git commit -m “branch1 first commit”
□ git log --oneline
□ git checkout master
□ git log --online
□ ls
□ git branch
□ git checkout branch1
□ cat >secondfile → add content
□ git checkout master
□ ls
□ git checkout branch1
□ git add .
□ git commit -m “branch1 2nd commit”
□ git log --oneline
□ git checkout master
□ ls
□ git log --oneline

## Git Merge :

• We can not merge branches of different repositories
• We use pulling mechanism to merge Branches

*Commands for Merge in Lab :*

□ git branch
□ ls
□ git log --online
□ git merge branch1
□ ls
□ git log --oneline
□ git push origin master

## Git Conflict :

• When same file having different content in different branches,if we do merge,conflict occurs(Reslove COnflict then add and Commit)
• Conflict occurs when we merge branches.

*Commands for git conflict in lab :*

□ git branch

□ ls
□ cat >rajputfile → add content
□ git add .
□ git commit -m “1st commit before conflict”
□ git checkout branch1
□ cat >rajputfile → add content
□ git add .
□ git commit -m “Commit from branch1”
□ git branch
□ git checkout master
□ git merge branch1
□ vi rajputfile → press i → modify content → press esc → enter :wq
□ git status
□ git add .
□ git commit -m “commit after resloving conflict”
□ git log --oneline

## Git Stash :

• Suppose we are implementing a new feature for our product,our code is in progress and suddenly a customer escalation comes because of this,we have to keep aside our new feature work for few hours
• We cannot commit our partial code and also cannot throw away our changes.So we need some temporary storage,where we can store our partial changes and later on commit it
• To stash an item (only applies to the modified files not new file)

*Commands for Stashing :*

□ git stash → To stash an item
□ git stash list → To see stashed items list
□ git stash apply stash@{0} → To apply stashed items
□ Then we can add and commit
□ git stash clear → To clear the stash items

*Commands of git stash in Lab :*

□ touch demofile
□ git add .
□ git commit -m “demofile”
□ git branch
□ vi demofile → press i → add content → press esc → :wq
□ git stash
□ vi demofile → returns empty file
□ cat demofile
□ git stash
□ git stash list
□ vi demofile → press i → add content → press esc → :wq

□ cat demofile
□ git stash
□ git stash list
□ cat demofile
□ git stash apply stash@{1}
□ cat demofile
□ git add .
□ git commit -m “First code done”
□ git stash apply stash@{0}
□ vi demofile → press i → modify content → press esc → :wq
□ git add .
□ git commit -m “Second code done”
□ git status
□ git log --oneline
□ git stash list
□ git stash clear
□ git stash list

## Git Reset :

• Git reset is a powerful command that is used to undo local changes to the state of a git repo

*Commands for git reset :*

□ git reset <file name>
□ git reset .
□ git reset --hard → To reset the changes from both staging area and working directory at a time

*Commands for git reset in Lab :*

□ git branch
□ cat >testfile → add content
□ git add .
□ git status
□ git reset .
□ git status
□ git add .
□ git status
□ git reset --hard
□ git status

## Git Revert :

• The revert command helps us undo an existing commit
• It does not delete any data in this process.Instead, git creates a new commit with the included files reverted to their previous state.So our version control history moves forward while the state of our file moves backward

*Commands of git revert in Lab :*

□ sudo su
□ cd singaporegit
□ ls
□ git status
□ git log
□ cat >revertfile → add content and save
□ git add .
□ git commit -m “Code1”
□ cat >>revertfile → add content and save
□git add .
□ git commit -m “Code2”
□ git log --oneline
□ git revert <Commit-Id>
□ modify commit message using vi and save
□ ls
□ cat revertfile
□ git log --oneline

**How to Remove Untracked File :**

• git clean -n → (dry run)
• git clean -f → (forcefully)

*Commands to remove untracked file in lab :*

□ touch filex filey filez filexx filexy filexz
□ git status
□ git clean -n
□ git clean -f
□ ls

**Tags :**

• Tag operation allows giving meaningful names to a specific version in the repository

*To apply Tags :*
• git tag -a <tag-name> -m <message> <commit-id>

*To see list of Tags :*
• git tag

*To see particular commit content by using Tag :*
• git show <tag-name>

*To delete a tag :*

• git tag -d <tag-name>

*Commands to use in Lab :*

□ git log --oneline
□ git tag -a important -m “This is Very imp commit” <commid-id>
□ git tag
□ git show imporatnt
□ git tag -d important
□ git tag
□ git log --online
☑ git tag -a important -m “This is Very imp commit” <Different commid-id>
□ git tag
□ git log --online

**Github Clone :**

• Open github website
• Login and choose existing repository
• Now,go to our linux machine,and run command :
□ git clone <url of github repo>
• It creates a local repo automatically in linux machine with the same name as in github account

*Commands to use in Lab :*

□ git clone <url of github repo>
□ ls
□ cd <name of cloned repo>
□ touch fileyy
□ git add .
□ git commit -m “New commit”
□ git push origin master