**GIT**

* It is a version control system or source code management
* It is used to track the changes in files.
* It will maintain multiple versions of the same file.
* It is platform independent
* It is free and open source
* They can handle larger projects efficiently.
* They save time and developers can fetch and create pull requests without switching.

SCCM RCS CVS SVN GIT

1972

2000 2005

To track Track multile files Track multiple Track multiple Distributed vcs

One file but not directories files& dir files& dir

**Free**: GIT, SVN

**Paid**: Bitbucket, P4, Stash

**Git stages**

**Working directory**

* In this stage git is only aware of having files in the project.
* It will not track these files until we commit those files.

**Staging area**

* The staging area is like a rough draft space, it’s where you can git add the version of a file or multiple files that you want to save in your next commit.
* In other words, in the next version of your project.

**Repository**

* Repository in git is considered as your project folder.
* A repository has all the project-related data.
* It contains the collection of the files and also history of changes made to those files.

**Types of repos**

**Local repo**: The local repository is everything in your .git directory. Mainly what you will see in your local repository are all of your checkpoints or commits. It is the area that saves everything (so don’t delete it)

**Central repo**: It will be present in the Github where you can share all your files. You need to add and commit your files before you push in Github.

**Remote repo**: It will be present in remote hosts where you can share all your files to remote.

W **Working Directory**

**git add**

**Staging area/Index area**

**git commit**

**Repository**

**How to configure git with github**

**To add user in git bash**

#git config –global user.name “venu”

#git config –global user.email [pvenu9390@gmail.com](mailto:pvenu9390@gmail.com)

**To list all config file**

#git config –list

**To check more information of command**

# git <command> --help

#git help <command>

==============================================================================

.**git** directory will track our entire projects- what change we made, who made the change and when etc

==============================================================================

Whenever you clone a projects, by default you get master branch

To clone the project from github

#git clone <repo url>

Do some changes and push it back to remote system, first we need to check the status

#git status

#git add <changed file name>

#git commit -m “few changes happened”

#git push origin master

**New Project**

* Create a git repository in github account
* Write a program code in our local machine
* First thing first do **git init** in git bash, it will create a .git directory in your project.
* **#git init**
* Then check **git status** once
* It will show the all untracked files.
* Then use **git add .** 🡺 it will change the state of files from untracked to tracked state
* Then commit all tracked file by using **git commit -m “<message>”**
* To link the server to our local machine by using below command

**#git remote add origin <git repo url>**

* Then push the code to your repo

**#git push -u origin master**

**Git Status Lifecycle**

**Untracked** : means git doesn’t know about these files.

**Tracked**: The know files for git. In tracked, files will be in three different states.

1. Unmodified--- file is ready to push.
2. Modified- if you did any changes before commit it will come to modified state
3. Staged—file is ready to commit.

If you want to see git status in short form use

* **#git status -s --short**

?? 🡺Untracked

A 🡺 Staging area

M 🡺 Modified files

**Git ignore file**

* Create a .**gitignore** file in your project. If you don’t want few files use some rules
* #Comments or blank lines
* Specific file: **intro.html**
* File pattern: **\*.txt**
* Folders**: <foldername>**
* Remember few file: **!main.txt**

**Git diff**

It will show the content difference between the working directory file and staged area file.

If you want to see content in the staging area file.

**#git diff –staged**

**#git diff –cached**

**Git Commit—commit is nothing but a snapshot**

#git commit ------- it will open editor prompt

#git commit -v ------It will show more information about file, what changes happened.

#git commit -m “<message>”

**Git log**

#git log ----- will show when it is committed

#git log -p ----- will show the difference between commits.

#git log -3 ----- it will display last 3 commits.

#git log –pretty=oneline ------ it will display in single line

#git log –pretty=short

#git log –pretty=full

#git log –pretty=fuller

**Git Branching**

* Pointer to one of the commit.

**Master branch---** it is point out recent commit.

**Head**--- represents currently working branch.

**Creating branch**

#git branch <branch name>

**Create branch name and switching same time**

#git checkout -b <branch name>

**If you want to change from one branch to another branch**

#git checkout <branch name>

**If you want to see all branches commits**

#git log --oneline --all

**If you want to see all branches commits in graphical mode**

#git log –online --graph –all

**How to merge the feature branch changes to master**

* First and fore most you must to checkout to master branch

**#git merge <branch to be merge> master 🡺 it is a fast forward merge**

**To check which branch you merged and which branch not**

#git branch –merge

**To check which branch you are not merged**

#git branch –no-merge

**To check all branches**

#git branch -a

**To check the remote server name**

#git remote

**How to delete branch**

#git branch -d <branch name>

**Merge Conflicts**

* If two changes(two developers) happened in same line and you are trying to commit

**Fetches and pull**

**Fetch**--- it will bring the changes from remote server, if you want that changes you need to merge manually.

**#git fetch origin master**

**#git branch -a**

**Pull**- it will bring the changes and merge it into our project file

**#git pull origin master**

**PUSH**

Before doing git push, first we need to do git pull and get the all updates from remote server and then do push.

#git push origin master

**P4merge tool**

**Download and install p4merge tool and config the diff and merge**

**Diff Tool---- git difftool**

#git config –global diff.tool p4merge

#git config –global difftool.p4merge.path”Path to p4merge”

#git config –global difftool.prompt false

**Merge Tool------ git mergetool**

#git config –global merge.tool p4merge

#git config –global mergetool.p4merge.path”Path to p4merge”

#git config –global mergetool.prompt false

**Alias**

#git config –global alias.<replace command> “original command”

**Rebase**

Merge and rebase will do the same work but the difference is the commit history

#git rebase <branch name>

After resolve the rebase conflicts

#git rebase --continue

**How to check what is happened in particular commit**

#git show <commit ID>

**Stashing**

Stashing as same as recycle bin in windows

#git stash (staged files moved into stash)

**If you want to see stash list**

#git stash list

**If you want to bring back from stash**

#git stash apply ---- recent moved file will come back to staging area

**If you want to delete from stash**

#git stash drop

**If you want particular stash file**

#git stash apply stash@{1}

**If you want to do apply and drop at the same time**

#git stash pop

**To include untracked files to stash**

#git stash -u

#git stash branch <branch name>

**Clean**

Clean will apply only on untracked files, it will not work on tracked file

#git clean -f ---- by default it will delete only file

#git clean -f -d ---- it will delete directory

#git clean -f -d -x ---- it will delete files and directories present in .gitignore file

#git clean -f -n ---- it will give warning bell to remove file

**Tagging--- tagging means marking, here we are marking particular commit**

**To check previous tags**

#git tag -l

**To create tag**

#git tag <tag name> 🡺 This is light weight tag

#git tag -a <tag name> -m “<comments>” 🡺 annoted tag

**How to compare two tags**

#git diff <tag one> <tag two>

**How to check when we release project**

#git log –online –graph –all

**How to give tag at particular commit**

#git tag -a <tag name> <commit id>

**How to delete particular tag**

#git tag delete <tag name>

**How to update the tag**

#git tag -a <tag name> -f <new commit id>

**How to push all tags to remote server**

#git push origin master –tags

**To list out all tags**

#git tag –list “tagname\*”