GitHub

**Git:**

Git is a **version control tool**(VCS) or **Source code management**(SCM) . It is developed by **Linus torvalds**(creator of Linux) in 2005. It is open source i.e ; freely available to use . It is most popular and most used version control tool.

* Git is a version control tool that performs all kind of operations to fetch data from the central server or push data.

**GITHUB:**

GITHUB is a central repository.

* GITHUB is used to hosting platform for version control .
* GITHUB is a company that allows you to host a central repository in a remote server.

**REPOSITORY:**

A directory or a storage space where your projects can live. It can be local or it can be a storage space on gather or other online host.

They are two types of Repositories:

1. Local Repository
2. Remote Repository or central Repository

**LOCAL REPOSITORY:**

1. Located in Local machine
2. Only admin of the machine can work with this repo.
3. Resides as a .git folder inside your project’s root.

**Central Repository:**

1. Located in remote Repository (gather).
2. Exclusively consist of .git repository folder.
3. It is meant for team to share and exchange data.

VERSION CONTROL SYSTEM:

To maintain multiple versions of a single file or directory

\*It is a tool that helps a software team manage changes to source code overtime.

\* In Git, version control software keep tracking of every modification done in the code in a special kind of database.

\* If any1 made a mistake we can easily fix that by using rollback or revert.

Version control system tools:

Free tools:

\*GIT

\*SVN

Paid tools:

\*STASH

\* P4

\* CLEAR CASE

\* BIT KEEPER

Why VCS:

To track different versions of a file or directory

What type of files tracks by VCS?

\*All Text based files.

What type of files not tracked by VCS?

* Videos and images

**HISTORY OF VCS:**

**SCCM:** To track only one file, 1972. Allows one user

**RCS:** To track multiple files not directories. Allows one user.

**CVS:** To track multiple files and directories. Allows one user

**SVN (sub version):** To track multiple files and directories. Allows multiple user

**GIT:** Distributed version control. Allows multiple user.

|  |  |
| --- | --- |
| **SVN** | **GIT** |
| 1. By using SVN , Every team member will connected to the server.  * Server = client based  1. SVN always need to connect internet. 2. Memory space is high. 3. Centralized version control.   Ex: it has only one server for all the developers.  5 . Back-up need to take an server | 1. By using GIT , Each and every developer has its own local system.  * Every project is independent   2.GIt want net, when we want to move the code to repo.  3. less memory.  4. Distributed version control.  Ex: every one has local system , whenever we complete the code we can push.   1. Back-up need to take from client system |

***GIT COMMANDS:***

***1.Git init:***

Git init command is used to install git on your local machineand used to create a new git local repository(directory/folder) or reinitialize the existing one.

**Command: git init**

***2.Git status:***

Git status command is used to display the state of current working directory and the staging area.

\*Git that you want to updates a certain file in the next commit.

\*This command will helps us to untracked files which files are to added or commited

\*\* here tracked files in “Green color”

**\*\* Untracked files in “Red color”**

***3.Git add filename:***

This command is used to add the untracked files to staging area.

Git that you want to updates a certain file in the next commit.

***4.git add –A***:

This command is used to add multiple files at a time.

Explanation: git add -A

If we have folder name is new. In that folder we created 4 text files like ex1.txt, ex2.txt, ex3.txt, ex4.txt. If we are adding these four files at a time then we use this command.

**5.git log:**

This command is used to allow you to review and read a history of everything that happens to a repository.

**Command: git log**

\* if we give this command it will generate a **sha-1-hash code** or **commit id** in yellow colour. It is 14 digit code like

Commit id : b09c0b51c6b9deafb1878e9651d3a9b5db5aff95

***6.git show commit id:***

This command will show us to what data is in the txt file.

**Command:**

**git show b09c0b51c6b9deafb1878e9651d3a9b5db5aff95**

***7.Git commit:***

\*It refers to recoding snapshots(versions) of the repository at a given time.

\*Committed snapshots will never change unless done explicitly.

**Command: git commit –m “commit message”**

**\*\***  Whenever we are committing always we have to give a commit message. Because everyone can see who made all these changes and what exactly it is.

***8.Git commit –a:***

This command is used to multiple commits at a time.

Explanation:

**Command: git commit -a -m “adding 4 files together”**

***9.Git clone:***

This command is to download the project.

If you are new to join the project then we use git clone command to download all the project.

***10.git branch branchname:***

This command used to create branch .

**Explanation**: when we create branch then that branch contains all the files that were in the master branch.

**Command*: git branch firstbranch***

***11.git checkout branch:***

\*This command is used to switch the branch from one branch to another branch. i.e: master branch to first branch

**Command: git checkout firstbranch**

**o/p: switched master branch in to firstbranch.**

**Note:** If we create a new file in new branch i.e., edu4.txt.after that git add, git commit , then we give list **command : ls** then it will return all the files in the newbranch(first branch).

**Changing new brnch to master branch:**

\*If we want to change newbranch to master branch.

**Command: git checkout master**

**12.Merging:**

It is a way to combine the work different branches together.

Allows to branch off, develop a new feature & it come back in.

**Command: git merge firstbranch**

**Note:** If we use merging command then we can merge first branch and master branch . after completing merge , if we give ls command then edu4.txt also appears here.

\*\* If we want to change anything in the new branch file then we first convert the branch to newbranch after that we will pass commit command. But it will not add to master branch so we have to merge those files.

***Cat filename:***

This command is used to show what we have modified in the text file.

Command: cat edu4.txt

o/p: it will return hat we have change the edu4 file

**13.git pull:**

This command is used to pull all the new lines or the changed files from a central repository and it directly places them or connect them into a master branch.

\*\* **git pull = git fetch + git merge**

**14.git fetch:** this command is used to fetches all the new files or changed files from the central repo.

\*\* when we use git fetch it stores in different branch which is not connected to the current workflow. To overcome this we have to merge. Then only we see the changes in our local repo.

***14.Rebasing:***

This is another kind of merging .It is used to make a linear sequence of commits. merging and rebasing is very similar. Both combining files but in different ways.

\*\* main advantage of rebasing is to get much cleaner project history.

\*\* it is used to reduce the number of branches(if we have so many branches we get confused. Like it is merged or not)

***Parallel development:***

***Branches:***

Branches are pointers to a specific commit.

Explanation: if we made changes in main branch , then remember that main branch is the master branch. Master branch will contain all the code .

Ex: If we are working on the master branch and if we made a change then we have to decided to add some new feature on to it.

\*so if we want to work on new feature individually or interfere with the master Branch if we want to separate that time we can create branch from this commit

Branches are two types.

\*\* Local branch.

\*\* Remote Branch

**Remote Branch**: Branches that going to connect your branches from your local repository to our central repository.

**Local branch**: we only create a branch in workspace and that is going to work with the files in our local repository.

***Merging:***

***Rebasing***

**Development process:**

**Step1 :** We need to create a directory.

Explanation: we do all our work in this particular directory only.

**Command to create a directory: mkdir directoryname**

**Step2:**  Main purpose of using git

Explanation: in this step , we are going to track the above created directory.

Command

Step3: we need to initialize git

Explanation: if we want to initialize the git into current working directory.

Command to initialize git: git init

Step 4: 3 phases in git

Explanation: after initializing directory , that directory automatically background creates 3 phases .

|  |
| --- |
| Local repo/ git directory |

Git commit

|  |
| --- |
| Staging area/cache area |

Git add

|  |
| --- |
| Working directory |

Working directory:

* In this state, git is just aware of the files in the project
* It doesn’t track the files yet.
* If we want to track these files . we have to commit these files by adding the files to the staging area . this brings us to the next state in git-life cycle

` command to enter the staging area: git add filename

Staging area:

* While we are in working directory, we have to select the files that have to be tracked by git.
* Why we do this ? why don’t e track everything in the project?

Bcz, some files in the project like class files, log files , results files and temporary data files area dynamically generated. it doesn’t make sense track the versions of these files.

* Where as the source code files, data files, configuration files and other project files contain business logic of the application . these files area tracked by the git are thus need to be added to the staging area.
* **Command to enter local repo: git commit – m “commit message”.**

**Local repo:**

Why all these 3 steps we use?

If we make changes to any file , if you need older version then we can go back to git revision to get that version bcz GIT maintain revisions.