**GIT AND GITHUB**

Git is a distributed version control system (VCS) commonly used for tracking changes in projects.

Types of VCS are,

1. Centralized version control system (CVCS) :

-The source code and its versions are stored on the central server.

-The files can be checked out of the server, made changes and then committed back to the central repository.

-Eg: Subversion (SVN)

2. Distributed version control system (DVCS):

- Every developer will have a complete copy of the source code repository including the history.

-Eg: Git

Some key concepts in Git include:

**Repository**: A repository, often referred to as a repo, is a collection of files and directories that make up a project. Git stores the complete history of the project, including all changes made over time, in the repository. The path of the remote repo will be /.git

**Commit:** A commit is a snapshot of the project at a specific point in time. It represents a set of changes made to the repository, such as adding, modifying, or deleting files. Each commit has a unique identifier and includes a commit message describing the changes.

**Branch**: A branch is a separate line of development within a repository. It allows developers to work on different features or bug fixes simultaneously without interfering with each other's work. Branches can be created, switched between, merged, or deleted.

**Merge**: Merging is the process of combining changes from one branch into another. It typically occurs when a feature or bug fix is completed and needs to be incorporated into the main branch, often referred to as the "master" or "main" branch.

**Pull Request**: A pull request is a mechanism for proposing changes to a repository. It allows developers to submit their changes to the project's maintainers for review before merging them into the main branch.

Git provides a command-line interface (CLI) and various graphical user interfaces (GUIs) to interact with repositories. Popular hosting platforms like GitHub, GitLab, and Bitbucket offer web-based interfaces and additional collaboration features for managing Git repositories.

**Installation**

1.Windows

-Can be downloaded from the google like any other applications.

-Git bash shortcut will be created and we can verify by running the git version command

**Stages of git**

**1. Working directory**

-It is the directory in the local laptop which has all the files of the project.

-It is not mandatory that all the files/folders in the local should be in the remote repo

**2. Staging area**

-It is referred as git index

-The files can be moved to staging area from the working directory with the git add command.

-The files in the staging area can easily be bought back to the working area without the creation of history.

**3. Git repository**

-Files in the staging area will be added to the .git repository with the command git commit -m “commit msg”

-The history can be tracked once the files are committed

**4. Remote repository (Github)**

-The commited files are pushed to the remote repo with the command git push <url of the repository>

Git aliases

-It is used to get the shortcut of the commands

-git config alias.<shortcut> <full command> --> This will specify the alias to a particular project

eg: git config alias.st status --> after this git st can be used to know the status

-git config --global alias.<shortcut> <full command> --> To make alias global for all the git projects

-To list all the alias created, open the .gitconfig in any editor and can also make any alterations required.

-Root-commit refers to the commit that was done first to the repository.

**Config**

- Used to add the information about the username and email info which is used for the identification purpose.

- git config --list --show-origin --> used to list the configurations

**Git Clone:**

-Cloning a repository creates a copy of an existing repository, including all its branches, commit history, and files in the local that can be modified and pushed back.

-It's the most common way to start working with a Git repository. When you clone a repository, you create a local copy

-Cloning is useful when you want to contribute to an existing project or collaborate with others. It sets up a local copy of the repository on your machine.

**Git Fork:**

-Forking a repository means creating a copy of an entire repository under your GitHub account. Forking is typically done on web-based Git platforms like GitHub, GitLab, or Bitbucket.

-When you fork a repository, you get a complete copy of the original repository, including all branches, commits, and files. However, the forked repository is independent of the original one, and any changes made to it do not affect the original repository.

-Forking is often used when you want to contribute to an open-source project. You create a fork, make changes to your forked repository, and then submit a pull request to the original repository to propose your changes.

-Forking is done through the web interface of Git platforms like GitHub. There is no specific Git command for forking a repository.

-If the changes made in the forked repo should be sent to the original repo, a pull request should be made.

**Tag**

- It is the ability to tag specific points in a repository’s history as being important like stable version, restore a perticular version.

- Need to checkout to the branch that has to be tagged and create tag using git tag <tag name> and created tag can be listed using git tag command.

- git push origin <tag name> to push a particular tag and git push --tags to push all the tags

- git push origin –delete/-d <tag name> --> to delete a tag in the remote repo

- git tag –delete/-d <tag name> --> to delete a tag in the local

**Reset**

-It is used to undo the changes without creating the history (I.e, not listed in the git log command).

- git reset HEAD <file name> --> moves the file from the staging are to working directory.

- To move back to a particular commit and undo all the changes after that commit, we should use git reset <commit ID of which we need to go> --> now the changes will remain only in the working directory.

- If we want to remove the changes in the working directory also then use git reset <commit ID of which we need to go> --hard

Revert

-It is used to revert the changes of a particular commit. This is similar to that of the reset but once revert is done, it will ask us to provide a commit msg which will be listed in git logs with a new commit ID.

-git revert <commit ID which is to be reverted>

-Revert is also used to revert a perticular commit in a sequence without affecting other commits.

**Rebase**

- Merge and rebase are used to intergrate the changes from one ranch to another.

- It rewrites the commit history unlike in git merge

**Cherry pick**

- Cherry picking is the act of picking a commit from a branch and applying it to another. git cherry-pick can be useful for undoing changes.

- First we should get the commit ID of the changes we need to merge, then checkout to the branch to which the change has to be added and do git cherry-pick <commitID>

- Can pick multiple commits to merge using git cherry-pick <commitID1> <commitID2> --> this will add the changes in the order of mention in the command.

- git cherry-pick –abort --> is used to cancel the cherry-pick command if there are any conflict.

**Stash**

- The git stash command enables you to switch branches without committing the current branch.

- The files must be added before performing git stash.

- git stash --> will stash all the files in the staging area into a single commit ID.

- git stash -m <name> --> the stash will be saved in the entered name and not with the commitID.

- git stash list --> will list the commit ID of the stash.

- git stash show --> will list all the files that are in the commit ID.

- git stash pop --> will add all the files in the stash stage to the branch from which we perform stash command.

**Fetch**

-It is used to get the changes from the remote repo to the local and the changes will not be automatically integrated to the working files. These files can be inspected and merged if required.

-

**Deleting file**

-The files in the work directory can be deleted directly using rm command as they are not yet tracked with git.

-Files that are tracked in git can be deleted using git rm <file name>

**Commands**

-/.git is the hidden folder in the local which contains the actual git repository

- git config --global user.name “User name” --> used to add the user name

- git config --global user.email “email address” --> used to add the email address of a user

- git config --global --list --> used to list all the users

- git init --> Used to initialize a repository in the current directory.

- git clone <github url> --> used for the first time if we are getting the copy of the repository in the remote to the local. Th url can be obtained from the remote repo

- git status --> used to know the status of the files/repos such as which is the current branch, is the files added/commited

- git push --> uploads all the local branch changes to the remote repo

- git push origin <branch name> --> pushes the local changes of a particular branch to remote repo

- git add <file name> --> add a particular file in the local to staging area

- git add . --> add all the files in the local to staging area

-git commit -m “commit msg” --> used to commit the staged files from staging area to git repository

-git commit -am “commit msg” --> used to add and commit the file

-git ls-files --> list all the tracked files. Can add the tags such as --stage, --modified, --deleted,

-git checkout -- <filename> --> will remove the recent changes made to the file in the working directory.

-git mv <old file name> <new file name> --> to change the file name. After changing the file name, it has to be committed again. If it is renamed using mv command, then in the status it will show that the file with old name is deleted and new nam is created.

-git log --> used to get the complete log details and to exit press q

-git log --oneline --> it is used to get the logs in a single line. It lists only the commit ID with the commit msg.

-git log <branch name > --oneline --> it is used to get the logs of a particular branch in a single line. It lists only the commit ID with the commit msg.