**Git & GitHub**

>> git - git is VCS(version control system) software designed to records changes made to the file over the period of time. It is centralize system have single server that have all versioned files and number of clients can take files from that central place. It allows multiple clients to work on same repository at same time.

>> github - it is a hosting platform to host our git repositories in cloud. This helps us to access the code from anywhere and also share the code around the world.

>> Three stages of git -

1. modified - it means you have made some changes to the file but have not commited it to the database.

2. staged - it means you have marked the modified file in its current version to go into next commit.

3. committed - it means file is safely stored in local database.

>> staging area - it is file contain in git directory, that stores information about what will go into your next commit. Its technical name in git is 'index' but phrase 'staging area' works as well.

>> repository area - it is git directory where the git stores the metadata and objects database for your project. This is most important part of the git and when you clone repository from another computer it is stored inside repository area.

>> git commands

mkdir directoryName --> to make new directory

ls --> list out all the folders present inside the directory

touch fileNameWithExtension --> to create new file with specified extension

git init --> to initialize the repository

git config --local user.email yourEmail --> to set email at project level

git config --local user.name yourName --> to set name at project level

git config --unset user.email --> to unset email

git status --> to check status of files inside directory

git add fileName --> to add specific untracked file into staging area

git add . --> to add all the untracked files into staging area

git commit -m 'message about commit' --> to stored all files from staging area into local database

git log --> to get all the history about your previous commits

git mv fileName1 fileName2 --> to change file name

git restore --staged fileName --> to restore file from staging area

>> branching in git -- when multiple clients are working on one project, if someone's code contains bug it may lead to collapse the entire project. So, every programmer when they are started to work on the new feature, they should create new branch and start working on our own branch.

git branch --> to show all branches

git branch branchName --> to create new branch

git checkout -b branchName --> create new branch and switch to newly created branch

git checkout branchName --> switch to other branch

git switch branchName --> switch to other branch

git switch -C branchName --> create new branch and switch to newly created branch

git branch -m branchNane --> to change current branch name (only change current using branch name)

git branch -D branchName --> to delete branch (we cannot delete current using branch)

git branch -r --> to list all remote branches

>> mearging two branches - when we want to merge code from other branch into master branch we use merge command.

There are two approaches in git merge -

1. fast forward - when we create a branch and without making any commit to this new branch we merge this branch into master branch

2. recursive method - when we create a branch and made commit to this new branch and merge this branch into master branch

git merge branchName --> to merge branch into master branch (we should do merging operation from master branch)

>> git conflict - A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts will most likely happen when working in a team environment.

git merge --abort --> to abort merging operation in the master branch

>> git rebase - rebase is often use as an alternative to merging. rebasing a branch is update one branch with another by applying a commits of one branch on top of the commits of another branch. The main difference between merge and rebase is merge preserves the history and rebase doesn't.

>> git amend - it lets you combine staged changes with the previous commit instead of creating an entirely new commit.

i.e. if we commit one file and after that make some changes in that file, so instead of making new commit we use amend to modify latest commit.

git commit -- amend

>> git reset - reset moves current branch and optionally copies the data from repositories to the staging or working area.

i.e. if we commit two files to the master one after another. again add some changes in second file and again make commmit. But, after some time we don't want that changes in our master branch. that time we use reset command

git reset sha1Value\_of\_initial\_commit --> this will moves branch to initial commit but instead of removing second file put this file in staging area

git reset sha1Value\_of\_initial\_commit --hard --> this will moves branch to initial commit and removes the second file as well

>> git stash - sometimes you want to switch the branches, but you are working on an incomplete part of your current project. And you don't want to make commit of half done work. git stashing allow you switch branchs without commiting the current branch.

Normally when you switch branch you will the code and switch to new branch. If you switch branch without commiting two things happens -

1. switch to the branch carrying the changes

2. git will not allow to switch the branch and ask to commit or stash the changes.

git stash --> to stash the current file

git stash list --> list all the files inside stash

git stash pop --> it will take the recently pushed file from stash, remove it from stash list and give it back to the branch

git stash save 'stashName' --> it will save the file in stash with specific name

git stash apply stashID --> it will pushed file from stash associated with stashID and without removing it from stash list give it back to the branch

git stash drop stashID --> it will delete file from stash associated with stashID

git stash clear --> clear complete stash

git stash branch branchName stashID --> creates new branch with stashed data

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

>> git clone - to clone remote repository into your local machine, we use git clone command

git clone <url>

git initialize the new repo on your local machine. Thing to remember is, when we are trying to do the git clone, we should not clone it in already existing repository.

>> SSH key - when working with GitHub repo, we often need to identify yourself to GitHub using username and password. An SSH key is alternate way to identify yourself that doesn't require to enter username and password everytime. Using SSH protocol, you can connect and authenticate to remote server.

git remote -v --> to check whether local repository is associated with any remote repository or not.

git remote add origin <url> --> adding remote repository to local repository

>> origin is the short name for the url.

git remote rename <oldName> <newName> --> rename the remote repository name

git remote remove <name> --> removing remote repository from local repository

>> git push - it is used to upload local repository content to a remote repository.

git push origin branchName --> to push the local branch and its content to remote repository

git push origin branchName:otherBranchName --> to push the local branch content to different remote repository branch

e.g git push origin new-branch:master --> it will push the code from new-branch from local repository to master branch from remote repository

>> -u option - -u option allows us to set the upstream of the branch we are pushing. Once we set the upstream for the branch, we can use the git push shorthand which will push our current branch to upstream.

git push -u origin branchName

**What is Git?**

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

Tracking code changes

Tracking who made changes

Coding collaboration

**What does Git do?**

Manage projects with Repositories

Clone a project to work on a local copy

Control and track changes with Staging and Committing

Branch and Merge to allow for work on different parts and versions of a project

Pull the latest version of the project to a local copy

Push local updates to the main project.

**Working with Git**

Initialize Git on a folder, making it a Repository

Git now creates a hidden folder to keep track of changes in that folder

When a file is changed, added or deleted, it is considered modified

You select the modified files you want to Stage

The Staged files are Committed, which prompts Git to store a permanent snapshot of the files

Git allows you to see the full history of every commit.

You can revert back to any previous commit.

Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

**Why Git?**

Over 70% of developers use Git!

Developers can work together from anywhere in the world.

Developers can see the full history of the project.

Developers can revert to earlier versions of a project.

**What is GitHub?**

Git is not the same as GitHub.

GitHub makes tools that use Git.

GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

In this tutorial, we will focus on using Git with GitHub.

**What is a Git repository?**

A [Git repository](https://bitbucket.org/product/code-repository) is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.

**Initializing a new repository: git init**

To create a new repo, you'll use the git init command. git init is a one-time command you use during the initial setup of a new repo. Executing this command will create a new .git subdirectory in your current working directory. This will also create a new main branch.

**Git Ignore**

When sharing your code with others, there are often files or parts of your project, you do not want to share.

Examples : log files, temporary files,hidden files,personal files etc.

Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

Git will not track files and folders specified in .gitignore. However, the .gitignore file itself IS tracked by Git.