GIT

GLOBAL INFORMATION TRACKER

- Git is used to track the 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 e ciently.
- It is 3rd generation of vcs.
- it is written on c programming
- it came on the year 2005

WORKING DIRECTORY:

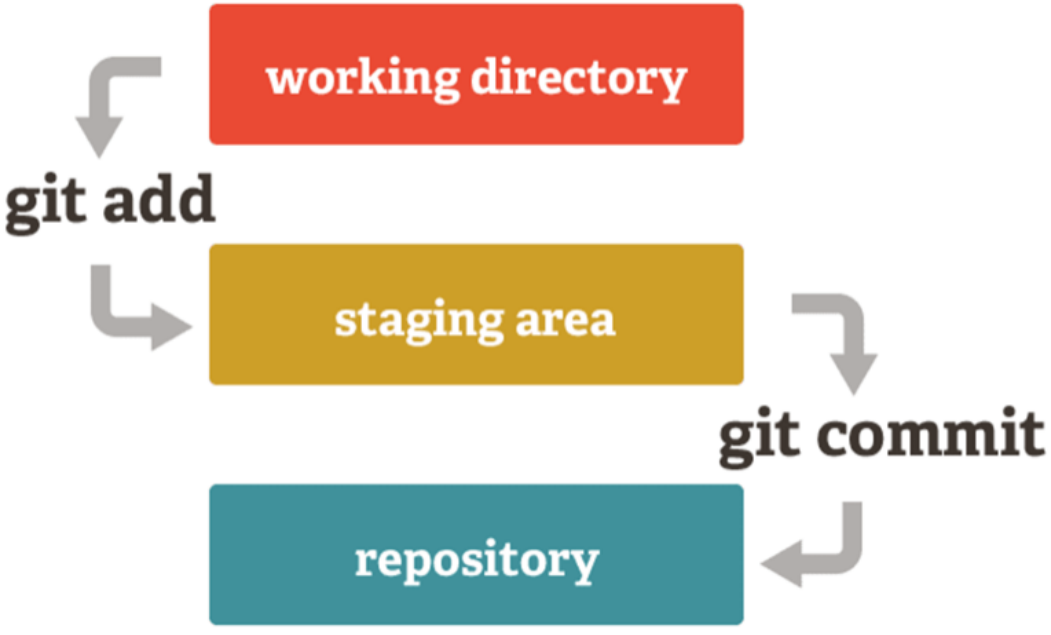
- In this stage git is only aware of having files in the project.
- It will not track these files until we commit those files.

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 REPOSITORIES:

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).

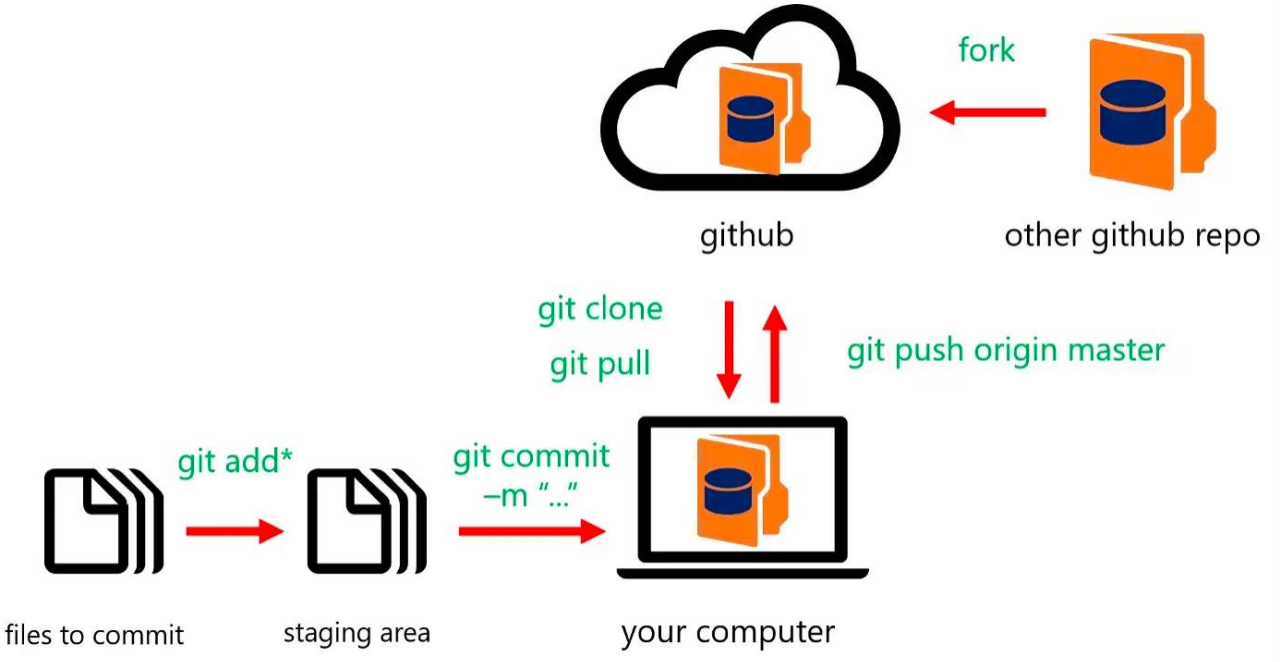
REMOTE REPO:

The remote repository is a Git repository that is stored on some remote computer.

GIT ALTERNATIVES:

- GIT LAB
- SVN
- BIT BUCKET
- P4
- STASH
- HELIX

GIT WORK FLOW:



INSTALL GIT:

1. command to install git : **yum install git -y** **(yum: yellowdog updater modifier)**
2. To check the git version: **git --version**
3. to get empty repo **: git init .**
4. To track the file: **git add file_name**
5. To track the multiple files : **git add aws azure gcp**
6. all regular files: **git add ***
7. including hidden files: **git add .**

GIT ADD:

- Git add command is a straightforward command. It adds files to the staging area.
- We can add single or multiple files at once in the staging area.
- Every time we add or update any file in our project, it is required to forward updates to the staging area.
- The staging and committing are co-related to each other.

GIT COMMIT:

- It is used to record the changes in the repository.
- It is the next command after the git add.
- Every commit contains the index data and the commit message.

GIT STATUS:

- The git status command is used to display the state of the repository and staging area.
- It allows us to see the tracked, untracked files and changes.
- This command will not show any commit records or information.

GIT CONFIGURE:

if you want to give your username and E-mail id to those commits then

1. git config user.name "username" : command to set username
2. git config user.email "userxyz@gmail.com" : command to set mail-id

GIT LOG: is a command in the Git version control system that shows a history of commits made in a Git repository. It displays a list of past changes, including information like the commit message, author, date, and a unique identifier for each commit.

COMMANDS:

git log	used to see the history of the git
git log --oneline	used to see only commit ID's and messages
git log --pretty=oneline	Used to get only commit ID's and messages
git log -1	used to see the latest commit
git log -3	used to see latest 3 commits
git log --follow --all filename	used to see the no of commits for a single file
git log --graph --oneline --all	see all the history in graph

GIT AMEND: it is a Git command used to make changes to the most recent Git commit. It allows you to edit the commit message, add additional changes, or both.

COMMANDS:

git commit --amend -m "message"	used to change the commit message for a latest commit
git commit --amend --author "username <mail>"	used to change the author of latest commit
git commit --amend --no-edit	used to commit the changes with previous commit

GIT RESET: is a command in Git that allows you to move the HEAD and the current branch pointer to a specific commit, e ectively "rewinding" or "resetting" your project's state. It's commonly used to undo changes or to unstage commits.

COMMANDS:

git reset --hard HEAD~1	used to delete the latest commit along with the changes
git reset --hard HEAD~3	used to delete the latest 3 commits along with the changes
git reset --soft HEAD~1	used to delete only commits but not actions/changes
git reset --soft HEAD~3	used to delete only latest commits but not actions/changes

GIT REVERT: is a command that allows you to undo or reverse the changes made in a previous commit. It creates a new commit that undoes the changes introduced by the specified commit, e ectively taking your code back to a previous state without deleting commit history

git revert commit_id	used to delete a particular commit action and add a new commit for the change

GIT IGNORE:

- It will be useful when you don't want to track some specific files then we use a file called .gitignore
- create some text files and create a directory with "jpg" files.

GIT BRANCHES:

- A branch represents an independent line of development.
- The git branch command lets you create, list, rename, and delete branches.
- The default branch name in Git is master.
- allows you to work on di erent features or changes to your code independently, without a ecting the main or other branches.
- It's a way to organize and manage your code changes, making it easier to collaborate and maintain your project.

COMMANDS:

git branch	used to see the list of branches
git branch branch-name	to create a branch
git checkout branch-name	to switch one branch to another
git checkout -b branch-name	used to create and switch a branch at a time
git branch -m old-branch new-branch	used to rename a branch
git branch -d branch-name	to delete a branch
git branch branch-name deleted-branch-id	Used to get deleted branch id
git branch -D branch-name	to delete a branch forcefully

The -d option will delete the branch only if it has already been pushed and merged with the remote branch. Use -D instead if you want to force the branch to be deleted, even if it hasn't been pushed or merged yet. The branch is now deleted locally.

Now all the things you have done is on your local system.

GIT MERGE:

Git merge is a command used in the Git version control system to combine changes from one branch.

To merge: git merge branch_name

GIT CHERRY-PICK:

Git cherry-pick is a command in Git that allows you to take a specific commit from one branch and apply it to another branch. It's like picking a cherry (commit) from one branch and adding it to another branch, allowing you to selectively copy individual commits without merging the entire branch.

Command: git cherry-pick commit_id

GIT MERGE CONFLICTS:

GIT makes merging super easy!

CONFLICTS generally araise when two people two people have changed the same lines in a file (or) if one developer deleted a file while another developer is working on the same file!

In this situation git cannot determine what is correct!

Lets understand in a simple way!

cat>file1 : hai all

add & commit

git checkout -b branch1

cat>file1 : 1234

add & commit

git checkout master

cat>>file1 : abcd

add & commit

git merge branch1 : remove it

git di file1

vim file1

git add

git commit -m "final commits"

NOTE: Don't give file name on commit

Identify Merge Conflicts:

see the file in master branch then you will see both the data in a single file including branch names that are dividing with conflict messages

Resolve Conflicts:

open file in VIM EDITOR and delete all the conflict dividers and save it!

add git to that file and commit it with the command (git commit -m "merged and resolved the conflict issue in abc.txt")

GIT REBASE: Git rebase is a Git command used to incorporate changes from one branch into another. It allows you to re-organize and streamline the commit history by moving or combining commits from one branch onto another.

Command: git rebase-branch

GIT STASH: Using the git stash command, developers can temporarily save changes made in the working directory. It allows them to quickly switch contexts when they are not quite ready to commit changes. And it allows them to more easily switch between branches.

Generally, the stash's meaning is "store something safely in a hidden place."

COMMANDS:

git stash	to delete the changes permanently
git stash save "message"	to save the stash along with the message
git stash apply	to get back the data again
git stash list	to get the list of stashes
git stash clear	to clear all stashes
git stash pop	to delete the first stash
git stash drop	used to delete the latest stash
git stash drop stash@{2}	used to delete a praticular stash

GIT TAGS: it tags are markers or labels that you can place on specific commits (versions) in a Git repository. These tags provide a way to give meaningful names to important points in the project's history, such as software releases or significant milestones.

COMMANDS:

git tag	To see the list of tags
git tag tagname	To create normal tag
git tag -a -m "message" tagname	To create annotated tag
git tag -d tagname	To delete a tag
git show tagname	To see the details of the tag

GIT-HUB

- GitHub is a web-based platform used for version control.
- It simplifies the process of working with other people and makes it easy to collaborate on projects.
- Team members can work on files and easily merge their changes in with the master branch of the project.

COMMANDS:

git remote add origin repo-url	link local-repo to central-repo
git remove -v	used to get the linked repo in github
git push -u origin branch-name	push the code from local to central
git push -u origin branch-1 branch-2	used to push the code to multiple branches
git push -u origin --all	used to push the code to all branches at a time
git clone repo-url	used to get the code from central to local
git pull origin branch	used to get the chanes from central to local
git fetch branch-name	used to fetch the data from central to local
git fetch --all	used to fetch the changes from all branches in github
git merge origin/branch	used to merge the changes from cental to local
git push -u origin --delete branch-name	used to delete the github branch from local
git remote rm origin	used to unlink the github-repo
git remote rename old -link new-link	used to change the repo