In SDLC how Many Steps we have 7 Steps

Devt

Plan -> Technical Architects, Business Analysts

Code -> Developers will write the source code

Build

Test

Technical Architects will do analyse the requirements and plan the architecutrue (Blueprint)

Q) Why to source write the source code?

GIT :

=====

Global Information Tracker

It is a Version Control System (VCS) or

It is a Source Code Management (SCM).

It will keep the Source Code seperately for each version.

index.html

V-1 : 100 Lines ---> Store (Repo-1)

V-2 : 200 Lines ---> Store (Repo-2)

V-3 : 300 Lines ---> Store (Repo-3)

REPO or REPOSITORY:

-> It is a Folder where we can store our Source Code.

GIT:

----

-> 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 large projects efficiently

-> It is 3rd generation of VCS.

-> It is written on C Programming Language.

-> It is Invented by Linus Torvalds.

-> It came on the year 2005.

example:

index.html -> It is a basic file for every application.

V1 ----> index.html

V2 ----> index.html

V3 ----> index.html

Version Control System:

-----------------------

1) This is the process of maintaining multiple versions of the code

All the team members upload their code(check in) into the remote

version controlling system. The VCS accepts the code uploads from

multiple team members and integrates it so that when the other

team members download the code they will be able to see the entire

work done by the team

2) VCS's also preserve older and later versions of the code so that

at any time we can switch between which ever version we want

3) VCS's also keep a track of who is making what kind of changes

VCS:

CVCS: Centralized Version Controlling System

SVN - It can store code on a Single Repo.

DVCS: Distributed Version Controlling System

GIT - It can store code on a Multiple Repo.

CVCS: Centralized Version Controlling System

Here we have a remote server(code repository) into which all the team

members check in the code and all the features of version controlling

are implemented in this remote server

DVCS: Distributed Version Controlling System

Here we have a local repository installed on every team members machines

where version controlling happens at the level of individual team members

form where it is uploaded into a remote server where version controlling

happens for the entire team

Roll Back:

----------

Going back to Previous Version of Application.

**Setting up git on Windows**

-------------------------------

1) Download git from

https://git-scm.com/downloads

2) Install it

3) Open gitbash and execute the git commands

----------------------------------------------------------------------

Configuring user and email globally for all users on a system

git config --global user.name "madhan gce"

git config --global user.email "madhangceb1@gmail.com"

----------------------------------------------------------------------

Git Stages / Git Architecture

On the local machine git uses three sections

1. Working Directory

2. Stagging Area

3. Local Repository

**1. Working Directory**: Where we write our source code.

Working directory is the location where all the code is created

**Initially all the files present here are called as untracked files**

**2. Stagging Area**: we track files here.

Stagging area is the location where file indexing happens and it

is the buffer area of git and the files are called as **indexed files**

**3. Local Repository**: where we store tracked source code.

Local repository is where version controlling happens and the files

are called as **committed files**

--------------------------------------------

**Working with GIT or Basic Commands of GIT:**

--------------------------------------------

1. To initialize the working directory to accept

git commands.

**Open "Git Bash" cd Path\_of\_Working\_Dir**

**git init**

The above command will create a hidden folder called **.git**

where it stores some configuration files for maintaining the local repository.

2. To send file from working dir to stagging area

git add filename

To send multiple files into stagging area

git add file1 file2 file3

To send all files and folders into the stagging area

git add .

3. To unstage files

( bring file back from stagging to untracked section )

git rm --cached filename

(or)

git reset filename

4. To send files from stagging to local repository

git commit -m "Some Commit Message"

5. To see the status of the untracked and stagging section

git status

6. To see the total commits that are done in LR

git log

To see the commit history in a simplified format

git log --oneline

-----------------------------------------------------

.gitignore

This is a special file where we can store the private filenames.

Any filename that is mentioned in .gitignore will not longer be accessed by git

1. Create few files

touch file1 file2 file3 file4

2. Check the status of git

git status

It will show all the above 4 files as untracked file

3. Create a hidden file .gitignore and enter the above filenames

cat >.gitignore

file1

file2

file3

file4

To come out of cat command press ctrl+d (EOF)

4. Check the status of git

git status

It will no longer show the above created four files as untracked

-------------------------------------------------------------------------------

Git Branches:

-------------

Branching in Git

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

This is a feature of git using which we can create seperate branches

for different functionalities and later merge them with the main branch

also known as the master branch. This will help in creating the code in

an uncluttered way

1. To see the list of local branches

git branch

2. To see the list all branches local and remote

git branch -a

3. To create a branch

git branch branch\_name

4. To move into a branch

git checkout branch\_name

5. To create a branch and also move into it

git checkout -b branch\_name

6. To merge a branch

git merge branch\_name

7. To delete a branch that is merged

git branch -d branch\_name

This is also called as soft delete

8. To delete a branch that is not merged

git branch -D branch\_name

This is also known as hard delete

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

Note: Whenever a branch is create whatever is the commit history of the

parent branch will be copied into the new branch

Note: Irrespective of, on which branch a file is created or modified git only

considers form which branch it is commited and the file belongs to that

commited branch only.

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

Working on the Github

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

This is the remote repository into which the code is uploaded and

this process is called as checkin

1. Singup for a github account

2. Signin into that account

3. Click on + on top right corner

4. Click on New repository

5. Enter some repository name

6. Select Public or Private

7. Click on Create repository

8. Go to Push an existing repository from command line and copy paste the commands

Enter username and password of github

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

Downloading from remote git repository:

This can be done in 3 ways.

1. git clone

2. git fetch

3. git pull

git clone:

---------

This will download the entire remote repository from github into individual developers machine

irrespective of whether that code is already present or not.

syntax:

git clone remote\_repo\_url

git fetch:

----------

This will work only when there are modifications in the code between the local repository and remote repository

git fetch will download the modified files and place them on a remote branch. We can go to

that remote branch check if those changes are acceptable or not and if they are acceptable

we can merge them with master.

1. Create some mofications to any file on github.

open github.com

click on our remote repository

click on any file that we want to modify

click on edit icon

make some changes ---> click on commit changes

2. In git bash

git fetch

3. Check the modified file on master branch

we will not see any modifications

4. The modifications will be on the remote branch

Move to the remote branch

git branch -a

git checkout -b remotes/origin/master

5. View the modified file

if the changes are acceptable merge with master

git checkout master

git merge remotes/origin/master

git pull:

---------

This will also work only when there are modifications between the local repository and the

remote repository.

But it will directly merge the modified file with master branch.

1. Open github.com

2. Make some changes to a file --> commit changes

3. In git bash

git pull

The modified files can be directly seen on the master branch

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

Git Merge:

----------

Merging always happens bases on the time stamps of the commits

1. Create few commits on master

touch f1

git add .

git commit -m "a"

touch f2

git add .

git commit -m "b"

2. Check the git commit history

git log --oneline

3. Create a test branch and create few commits on it

git checkout -b test

touch f3

git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"

4. Check the commit history

git log --oneline

5. Go back to master and create few more commits

git checkout master

touch f5

git add .

git commit -m "e"

touch f6

git add .

git commit -m "f"

6. Check the commit history

git log --oneline

7. Merge test with master

git merge test

8. Check the commit history

git log --oneline

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

Git Rebase:

-----------

This is called as fastforward merge where the commits coming from a

branch are projected as the top most commits on master branch

1. Implement step1-6 from above scenario

2. To rebase test with master

git checkout test

git rebase master

git checkout master

git merge test

3. Check the commit history

git log --oneline

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

Git Cherrypicking

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

This is used to selectivey pick up certain commits and add them to the

master branch

1 On master create few commits

a--->b

2 Create a test branch and create few commits

git checkout -b test

a--->b--->c--->d--->e--->f--->g

3 To bring only c and e commits to master

git checkout master

git cherry-pick c\_commitid e\_commitid

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

Git reset

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

This is a command of git using which we can toggle between multiple

versions of git and access whichever version we want

Reset can be done in 3 ways

1 Hard reset

2 soft reset

3 Mixed reset

In hard reset HEAD simply points to an older commit and we can see the

data as present at the time of that older commit

1 Create few commits on master

a-->b--->c

2 To jump to b commit from c

git reset --hard b\_commit\_id

------------------------------------------------------------------

Soft reset will also move the head to an older commit but

we will see the condition of the git repository as just one

step prior to the c commit ie the files will be seen in the

stagging area

git reset --soft b\_commitid

--------------------------------------------------------------------

Mixed reset also moves the head to an older commit

but we will see the condition of git as 2 steps prior to the c commit

ie the files will be present in the untracked/modified section

git reset --mixed b\_commitid

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

Git Stash:

----------

1. To Stash all the file in the stagging area

git stash

2. To Stash all the files in the stagging area and untracked section

git stash -u

3. To stash all the files in the stagging area and untracked section & .gitignore

git stash -a

4. To see the list of stash

git stash list

5. To Unstash a latest stash

gir stash pop

6. To unstash an older stash

git stash pop stash@{stash\_no}

.gitignore is a hidded file of git which is used for storing the name of private/secret files.

Any file whose name is mentioned in .gitignore remains unaccessable by git.

But .gitignore itself can be access by git, which means .gitignore moves into the stagging

area, local repository and the remote git repository

To prevent this and make .gitignore hidden from git

git stash -a

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

Git sqaush

-----------

This is the process of merging multiple commits and making

it look like a single commit.This can be done using the git rebase

command

1. Create a commit history

a --> b --> c --> d --> e --> f

HEAD is pointing to f commit

Note: a commit is called as the "initial commit" and it cannot be

squashed

In the above scenario we can sqaush only a max of 5 commits

2. To squash

git rebase -i HEAD~5

This will open the top 5 commits in vi editor

For which ever commits we want to perform a squash opration

remove the word "pick" and replace it with "squash"

3. Check the commit history

git log --online

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

Git rebase can also rearrange the commit history order

1. Create a commit history

a --> b --> c --> d --> e --> f

HEAD is pointing ti f commit

2. To rearrange the commit history order

git rebase -i HEAD~5

Reaarange the commits in whatever order that we want

3. Check the commit history now

git log --oneline