**GIT**

**Git Book Link:**

https://git-scm.com/book/en/v2

**Git Cheat-Sheet Link:**

https://about.gitlab.com/images/press/git-cheat-sheet.pdf

Git is a distributed version control system created by Linus Torvald.

Difference between Centralized version control system and distributed version control system.

**CVCS (Centralized version control system):**

* In CVCS, a client needs to get a local copy of the source from the server, do the changes and commit those changes to the central source on the server.
* CVCS systems are easy to learn and set up.
* Working on Branches is difficult in CVCS. Developers often face merging conflict.
* CVCS systems do not provide offline access.
* CVCS is slower as every command need to communicate with the server.
* If the CVCS server is down, Developers cannot work.

**DVCS (Distributed version control system):**

* In DVCS, each client can have a local branch as well and have a complete history on it. The client needs to push the changes to the branch which will then be pushed to the server Repository.
* DVCS systems are difficult for beginners. Multiple commands need to be remembered.
* Working on branches is easier in DVCS Developer’s faceless conflict.
* DVCS system is working fine on offline mode as a client copies the entire repository on their local machine.
* DVCS is faster as most user deals with a local copy without hitting the server every time.
* If the DVCS server is down, a developer can work using their local copies.

**Stages of Git:**

1. Workspace / Working Directory
2. Staging Area
3. Local Repo

-**git init** 🡪 command to create local repo

-**add** command used to put files from workspace to staging area.

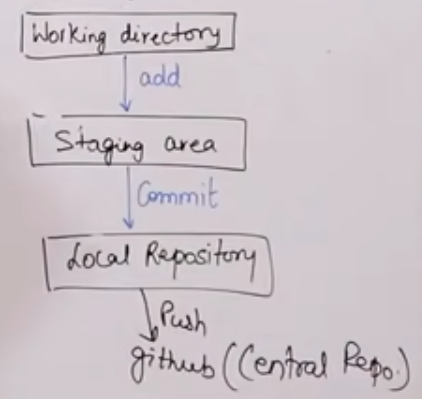
-**commit** used to put files from the staging area to the local repo. Every commit has a unique id and we can also add tags with the commit.

GitHub, GitLab, BitBucket, etc. are Central Repositories to store code.

-**push** used to put data into the central repo.

-**pull** used to get data from the central repo.

In other CVCSs, developers generally make modifications and commit their changes directly to the Repository but git uses a different strategy. Git does not trach each and every modified file whenever you do commit an operation git looks for the files present in the staging area only those files present in the staging area are considered for commit and not all the modified files.



**Commit:**

* Store changes in the repository you will get one commit-id
* It is 40 alphanumeric characters.
* It uses the SHA-1 checksum concept.
* Even if you change one dot, commit-id will get changed.
* It actually helps you to track the changes.
* Commit is also named as SHA1 hash

**Commit-id/Version-id/Version**

* Reference to identify each change.
* To identify who changed the file.

**Tags:**

* Tags assign a meaningful name with a specific version in the repository once a tag is created for a particular save, even if you create a new commit, it will not be updated.

**Snapshots:**

* Represents some data of a particular time.
* It is always incremental i.e it stores the changes (appended data) only not entire copy.
* Push:
* Push operations copies changes from a local Repository server to a Remote or Central Repo. This is used to store the changes permanently into the git Repository.

**Pull:**

Pull operation copies the changes from a Remote Repository to a local machine. The pull operation is used for synchronization between two repos.

**Branch:**

* When you create a branch all data present in the master branch at that time will be copied to the new branch and then you can work on the new branch separately and that will not affect your main branch.
* Branch marge will merge both branches’ codes.
* Creation of new branch is import if we don’t want to disturb our main branch work.
* Product is same, so one Repository but different task.
* Each task has one separate branch.
* Finally merges (code) all branches.
* Useful when you want to work parallelly
* Can create one branch on the basis of another branch.
* Changes one personal to that particular branch.
* Default branch is Master
* File created in workspace will be visible in any of the branch workspace until you commit once you commit then that file belongs to that particular branch.

**To Install Git**

-yum install git -y 🡪 to install git on Linux

-git --version 🡪 to check git version

- git config --global user.name “Your Name” 🡪 Set the name that will be attached to your commits and tags.

- git config --global user.email “you@example.com” 🡪 Set the e-mail address that will be attached to your commits and tags.

- git config --global color.ui auto 🡪 Enable some colorization of Git output.

-git config --list 🡪to see user configuration

-which git 🡪 to check where your git installed.

Start with Git

In the very first step you have to initiate git in the folder that you want to work

-git init 🡪 to tell that this direcotory is now git directory

You always work on your working directory

-git status 🡪 to check working direcotory status and files if red means the files are on working directory and if green means files and folders are on staging directory

-git add . 🡪 to add all things on staging from working directory

-git commit -m “first commit”🡪 to add your staging things to local directory

-git log🡪 to see all commits with commit ids

-git show <commit-id> 🡪 every commit has unique id you can check with (git log) command and by git show commit id you can check the changes you made on that commit.

If you want to add code on centeral repostitory you must have a centeral repository credentials then you can add by this command

-git remote add origin <central repo url> 🡪 to add central repo

-git push -u orign master 🡪 to push code in master branch in central repo

-git pull orign master 🡪 to pull code from repo

Ignore Files

To ignore files to add in git you have to create a file .gitignore and then you can add the extension of file or file name that you want to ignore

-vim .gitignore

=>\*.css

=>this.test

-git add .gitignore 🡪 all name present in the file will ignore

-git log -2 🡪 to check only last 2 commit

-git log --oneline 🡪 to check all commit logs in one line

Git Branch:

**﻿**

-git branch <name> --> to create new branch

-git checkout <branchname> --> to checkout from branch and go to other branch

-git branch -d <name> --> to delete branch

-git branch -D <name> --> to forcefully delete br

you can't merg branches of different repo

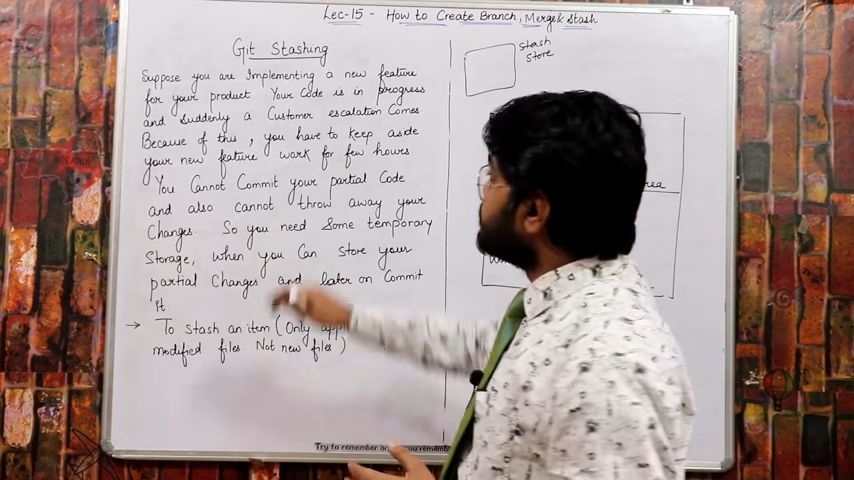
-git merge <branchname> --> to merg brancher

-git log --> to verify merge

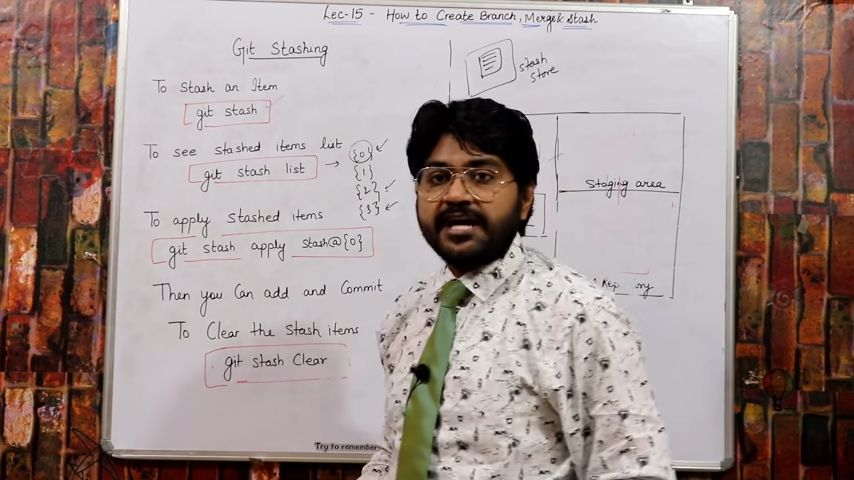
If you see conflict on branch merging you have to open file and change then merge again

Git Stashing

**﻿01:10:41﻿**



**﻿01:16:10﻿**



**﻿01:26:56﻿**

