1.**git init** ----> Will make this folder as a git repositry where tracking can happen

2.**git status** ----> to see the commits ar uncommits or branches details

3.**git add . or git add filenmaes** --->to commit files (Now changes will be tracked..these are staged)

Now the filesare stillin staging are not saved to git

4.**git commit -m "First Commit"**

-m --->Message First Commit is message for this commit

After this if you run git status everything will be clean as all things commited

**So if I want to change in the existing file**

a) Change

b) Stage

c) Commit

5. **Pushing local repository to github**

a) Get your repository address like "https://github.com/vishalgarg1994/test.git"

b) Add the local repository to command line **(git remote add origin "https://github.com/vishalgarg1994/test.git")**

c) Now you can push it (**git push origin master**) master--> branch name, origin ---> contains files from local repository

d) **git remote –v 🡪 To check, whether a repository is added or not**

6. **Cloning a repository**

**Git clone https://github.com/vishalgarg1994/test.git**

It will be cloned into the folder where you are presently in command line

Now it will be directly linked to the repository you are working on CLI (ORIGIN) … see 5(B)

If you make any changes here you can add,commit and directly push using origin

7. Now lets say we changed in clone and push to master, now we need to refresh main folder which is git repo.

**Git pull origin1 master (update repository)**

**Git Clone 🡪 New user came , wanted to clone full repository**

**Git Pull, 🡪 Already linked to main repository but need to get the latest changes**

**8. Unstage the Changes:**  You have done some changes and staged it, but now you want to unstage it to make some more changes---🡪 **git reset HEAD <file\_name>**

HEAD -🡪 A virtual pointer, always point to latest commit.

***Git Branches***

1. Adding a branch --🡪 git branch <name>
2. Listing branches -🡪 git branch
3. Deleting Branch 🡪 git -D <name>
4. Switching to new branch🡪 Git checkout <name>
5. When you are doing it for first time for new branch, the master branch’s last commit data will be copied to new one.
6. If you are doing it for existing branch, it will just switch
7. These branches will be isolate, changes in one won’t impact another
8. When you do a checkout ,you need to push all the changes to the branch brfore switching, else those will be lost(The solution to this is git stash)
9. **Git checkout <commit\_id>** : Could be used to see what happened till that commit point
10. **Git checkout HEAD~3 abc**.txt 🡪 Go to 3rd last commit nd extract the code for abc.txt. Now if you want to replace…> just stage and commit

***Git Stash***

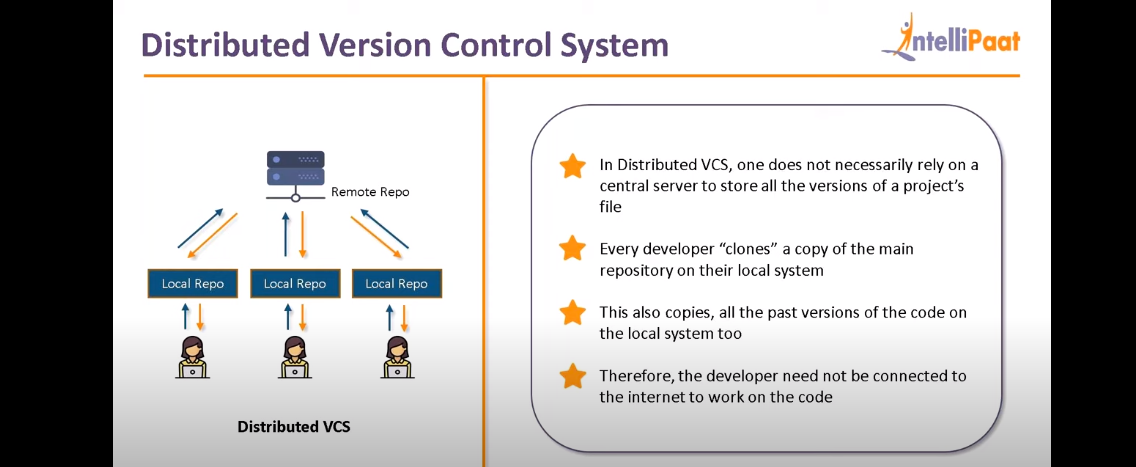
1. Save your work without committing the code
2. Can switch branches while saving your work but not committing
3. Issue: Let I have 2 branches(master,new) having 1 file 1.txt.
4. I am in new branch, added 2.txt and modified a.txt
5. When I switch to master both changes will be reflected in master as well because not committed.
6. So to avoid this type of confusion use **git stash –u** command
7. –u is required for untracked files, for files that are staged you don’t need it
8. This will make the things like no changes happened
9. You switch the branch, do your work, come back to your branch and use **git stash pop** to see all your changes back

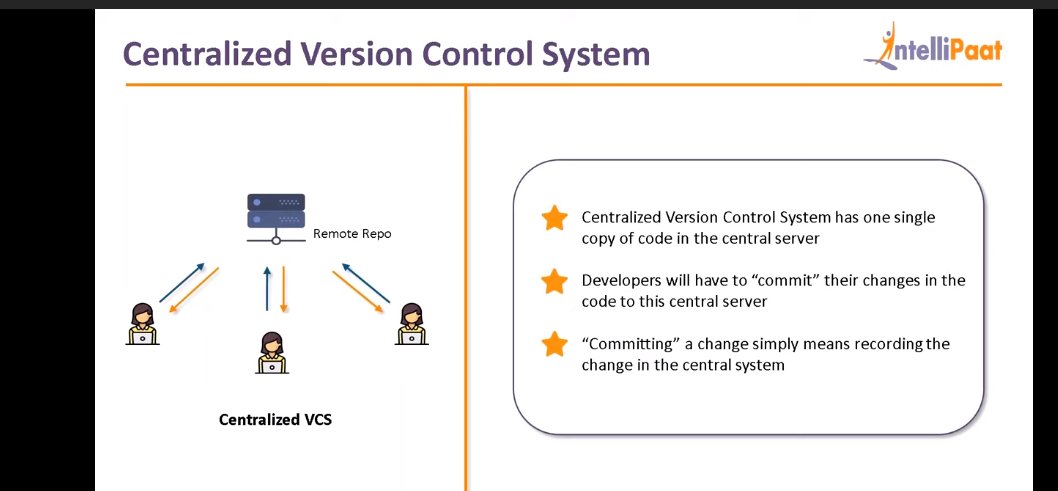
1. **Git Log** 🡪To see history of all commands (It will give history of current branch) Latest Commit will be present on the top **Git log -n 2 🡪**Latest 2 commits
2. **Git Log –stat 🡪** In every commit what has been changed, it will show
3. **Git Log –Oneline -🡪** Will tellthe changes done on every commit in a single line
4. **Git Log –author =”name” 🡪** Changes from a specific person
5. **Git revert <commit\_id>** -🡪 Return to a previous commit
6. **Git diff <commit\_id1: Older Commit> <commit\_id2 : Newer Commit>** : Differences b/w both commits
7. **Git diff HEAD .** 🡪 Will give you what happened from last commit till now on current brach. **Head points to latest commit**
8. **Git diff –staged 🡪**All staging files will be compared to the stuff stored in local .git(i.e. last commit)
9. **Git diff HEAD~n <file\_name>🡪** n is a number that will tell how many last commits will be compared, HEAD~3 will say, last 3 commits will be compared. Filename is optional, it tells only get the changes for this file not all files . Last 3 Means Latest with previous, Latest with one last version than previous

**If I am having 4 versions so 4vs3 and 4vs2**

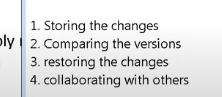
***Version Control Systems(VCS)***

1. Source code: In text format only, readable by humans
2. Only tracks source code, no jar,zip files etc
3. Only modified code is stored across versions so don’t take much storage
4. Repository 🡪 Like a folder containing all the codes of one application
5. Release is required when there are changes
6. So First Story board will be there what are new changes, what needs to be fixed and all those stuff
7. 2 Types, Centralized VCS(IBM BOX), Distributed VCS(Github)
8. Remote repo and centralized repo is same

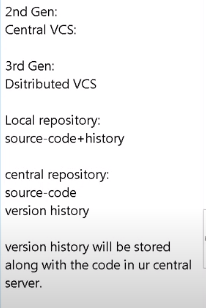




Usual things required from VCS

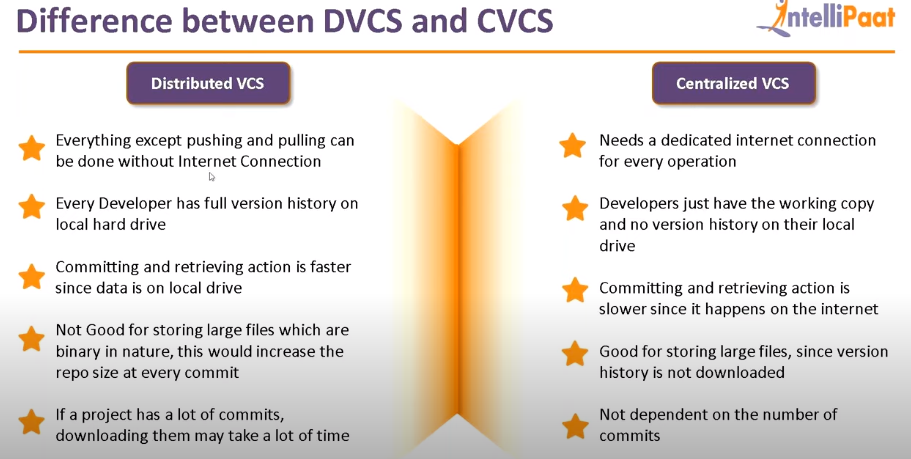


DCS Benefits

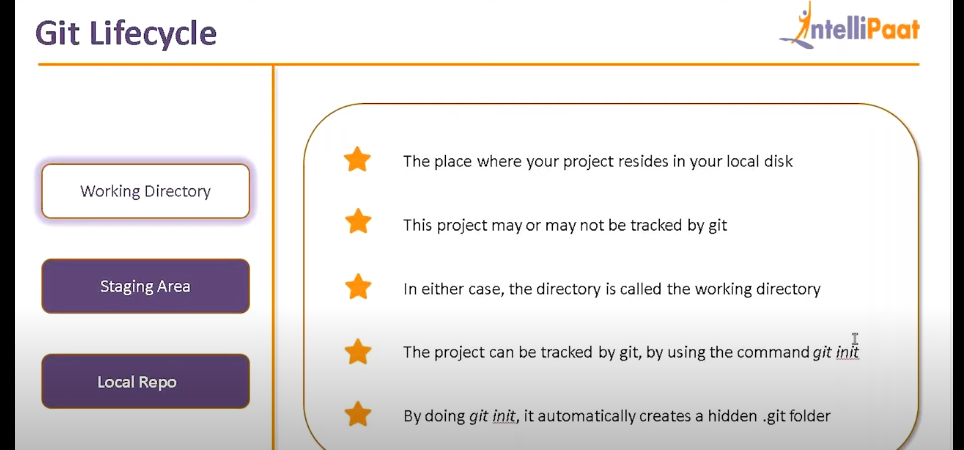


Other Benefits:

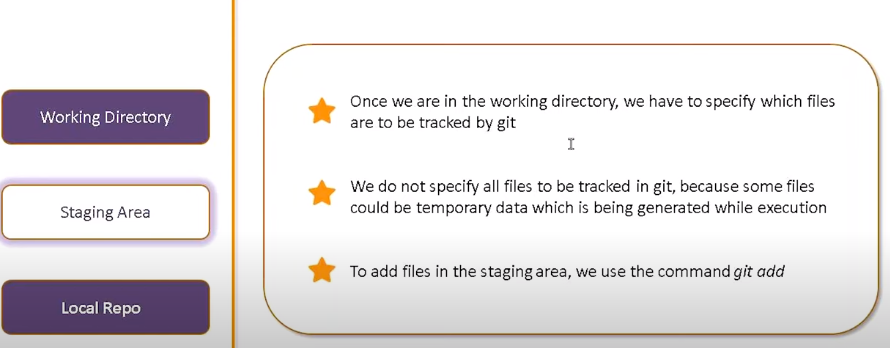
Lets say 2 developers(D1,D2) made clone of new branch and working on the same. D1 made some changes and push code to branch. Now D2 made changes and try to push the code but it won’t happen as D2 won’t have the latest copy of branch so it will be a conflict, Git will ask D2 to download latest version and make changes on top of that. So no conflicts



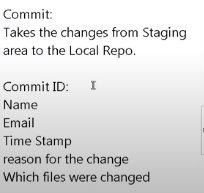
Git LifeCycle



In Staging area, Git gives us one more chance to review before committing







Whenever you are running a commit it will save above things and create a version in .git(hidden folder) and that version is with a commit id and ready to push to remote whenever you want

**Untracked files are those files that are never tracked by git**

**While Committing we need to give the defect id as a standard**

**Also try to use a single commit for all that changes you have done in multiple files so that it would be very much easy to track.**

**Git Status will compare all things from working directory to .git local repository and will tell you what is untracked or uncommitted and all that**

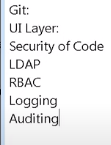
**So in the organization instead of github.com, it will be the server name where git is installed.**

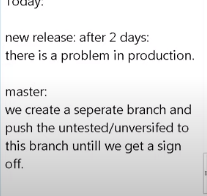
**Git is open source so won’t provide much security so organization buy the enterprise versions of it to track everything**

**Github -🡪 80% used**

**BitcBuket 🡪 15% Used**

**GitLab 🡪 5% used**





**Git->Settings->Branches To make sure no one can directly touch Master. You can set like someone will review it before moving and after review he can approve it.**

**Git Workflow**

**It will define a set of rules to follow while collaborating with others**

1. **Feature Release Workflow(For small projects, ops)**
2. **Git Flow workflow :complex situations(For Very big projects)**

**How Workflow works**

**1.New Task assigned**

**2. Go to central and Clone it. It will download the .git folder as well**

**3.Create a branch**

**4. Make changes\_>commit on current branch**

**5 Push this branch to remote**

1. **Now make sure your code is end to end tested**
2. **Pull request 🡪 It is a merge operation which also requires a review from a person. It means any changed thing for the files in new branch will be merged with the files in master branch after reviewal.**
3. **While raising a pull request you type in some message like,hi…please approve it to merge to master. This approval is usually done by release manager**
4. **When everything changes are in master you can deploy that**