***Git - Version control system / Source control management system***

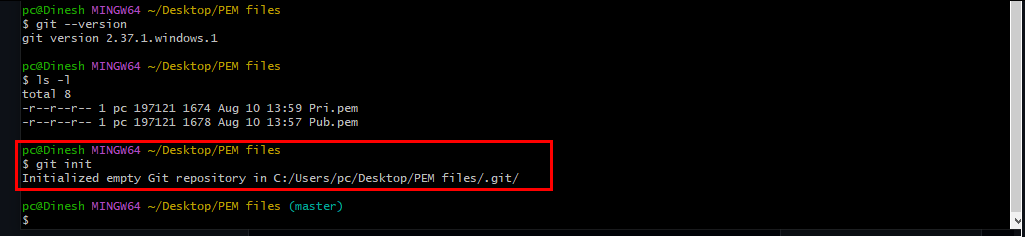
* It will track all the changes in the repository.
* Repository has all files and folders of a project.
* Git has its own file structure.
* Every file is stored in a separate document. It has a unique hash value.
* Git mostly works based on these hash values like key-value pairs which git maps to.
* Special file system

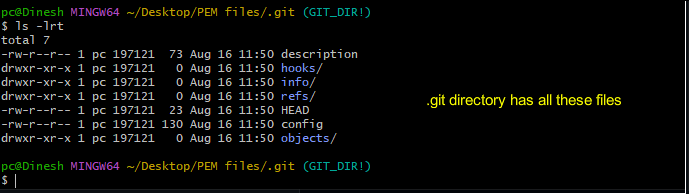
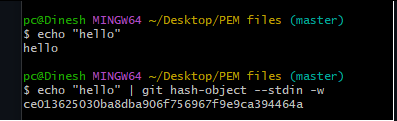
***Git-hub:***

* Repository Hosting Service
* Distributing the repository to multiple people.
* To backup the repositories.
* Giving access to people within the team.
* Can also host Websites.

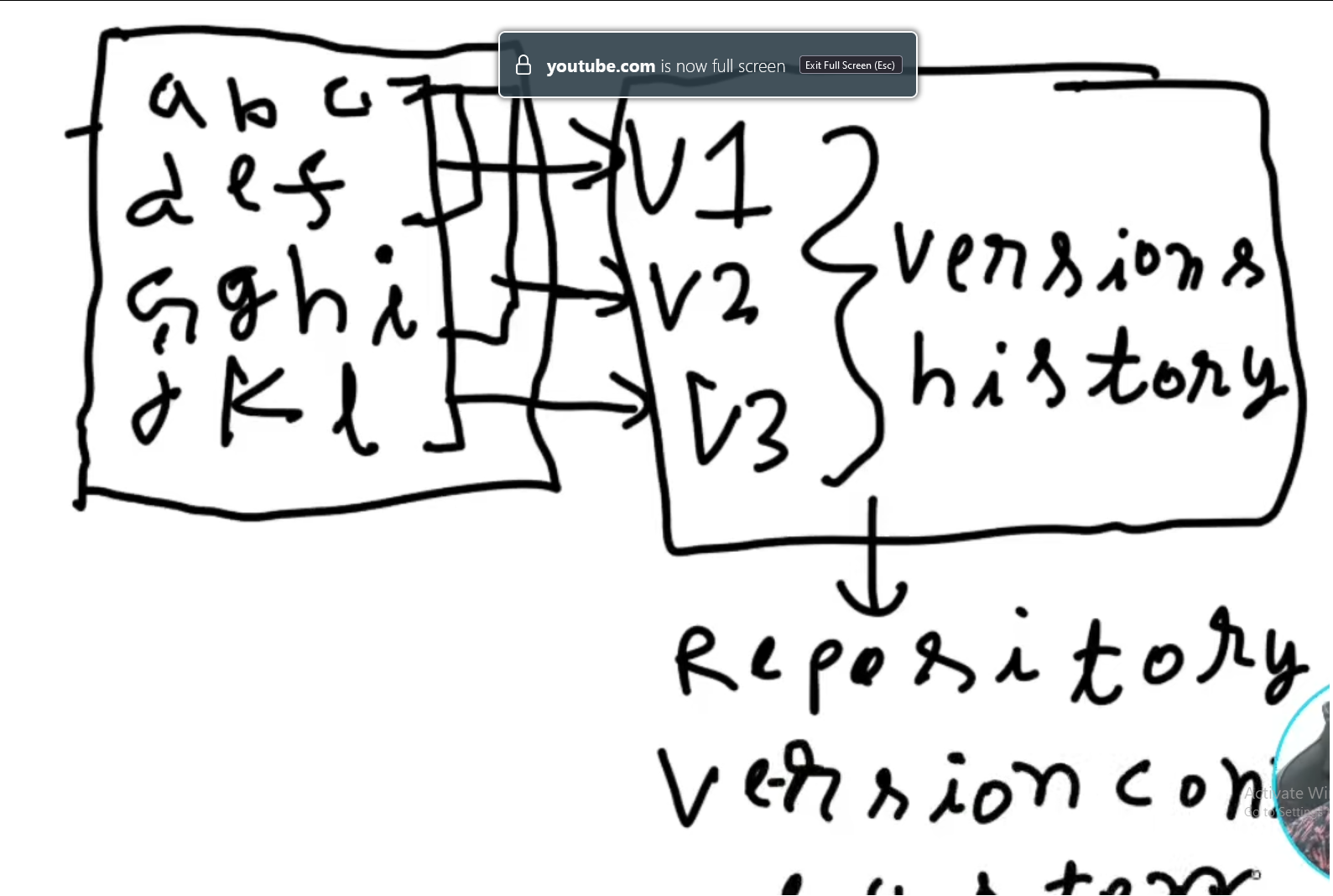
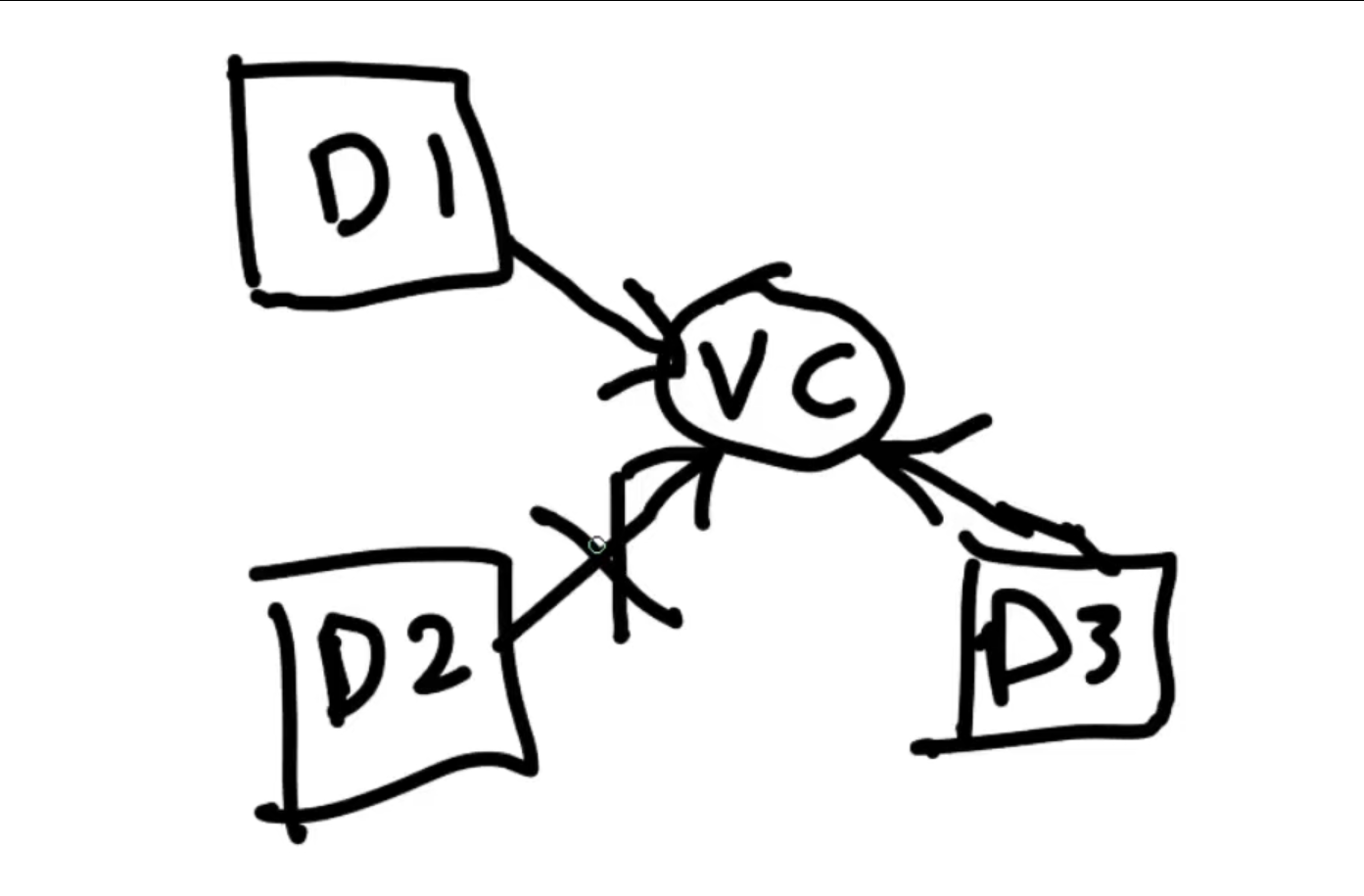
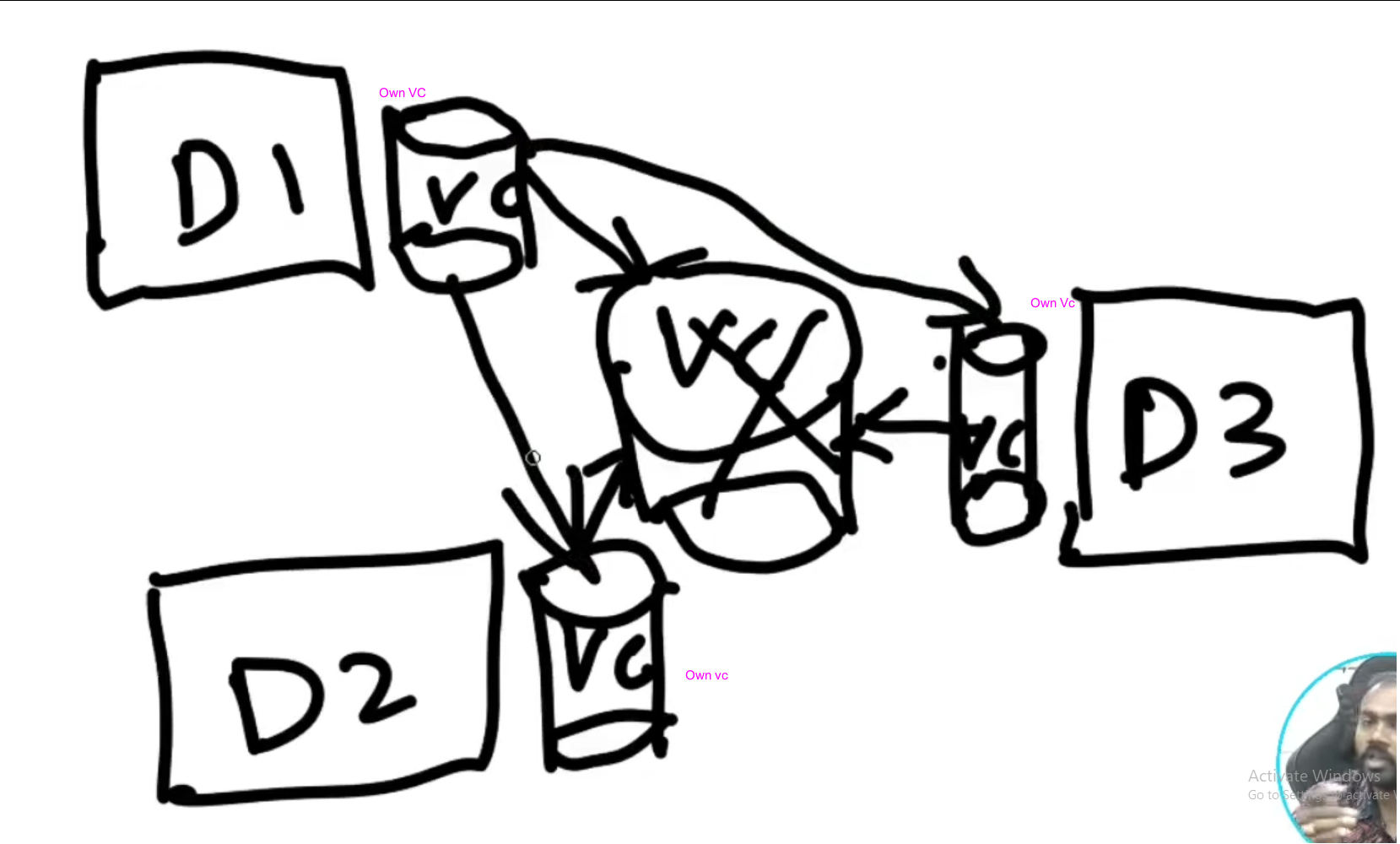
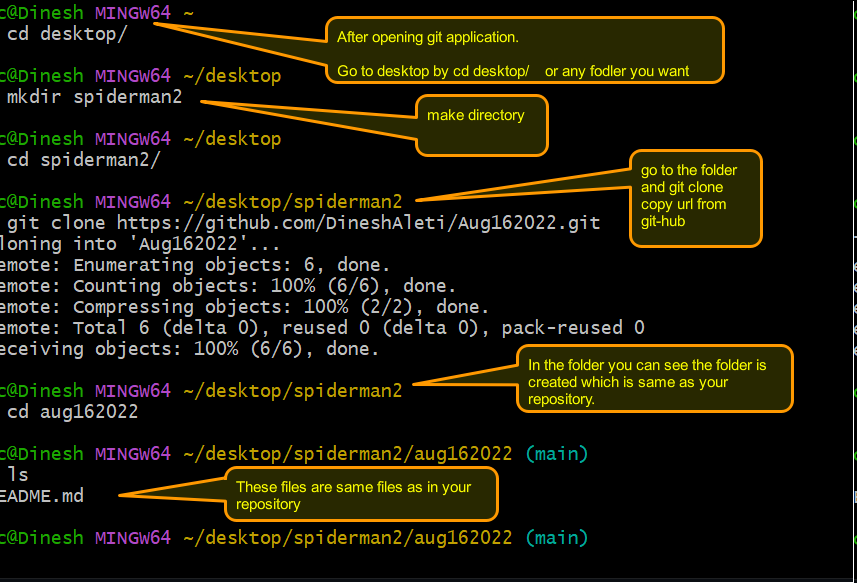
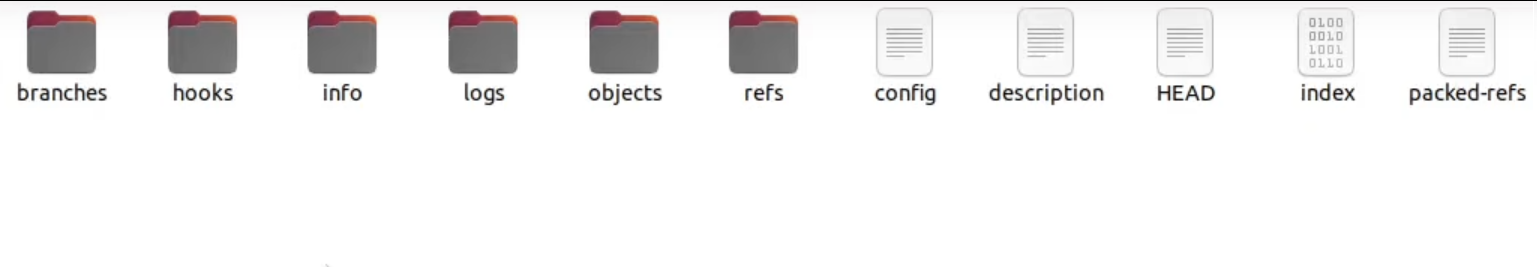
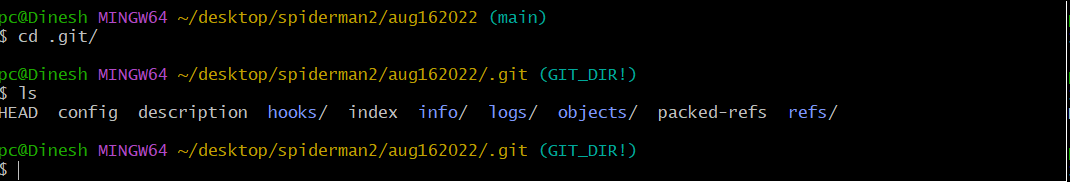
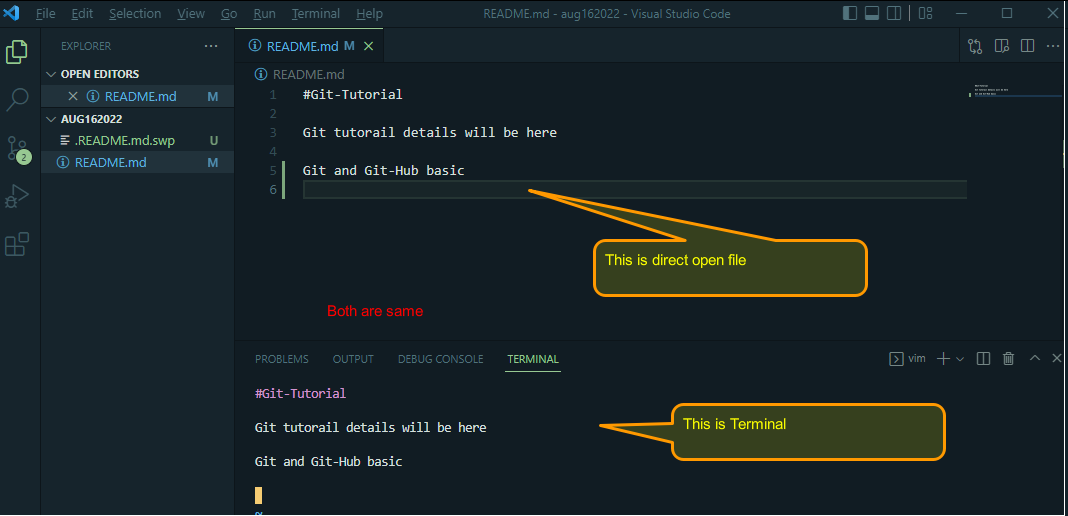
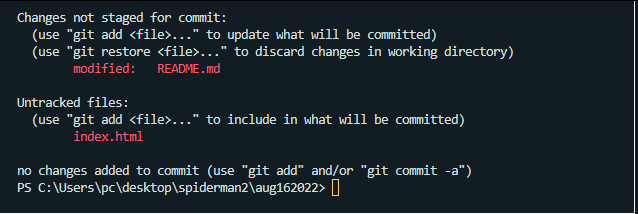
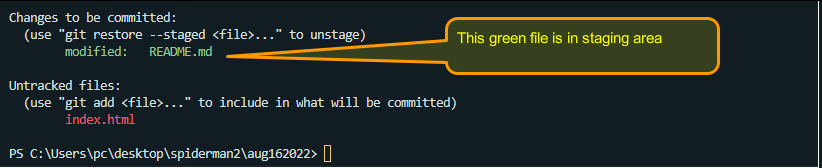
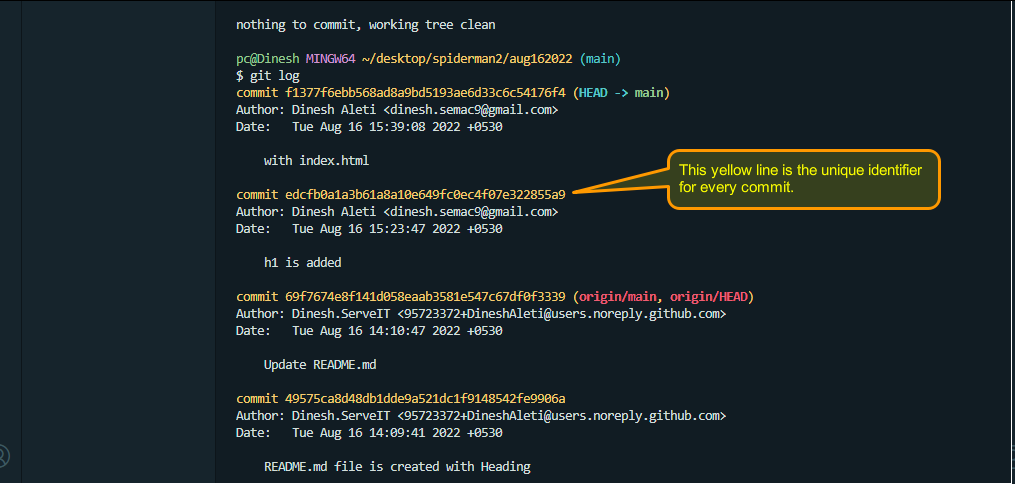
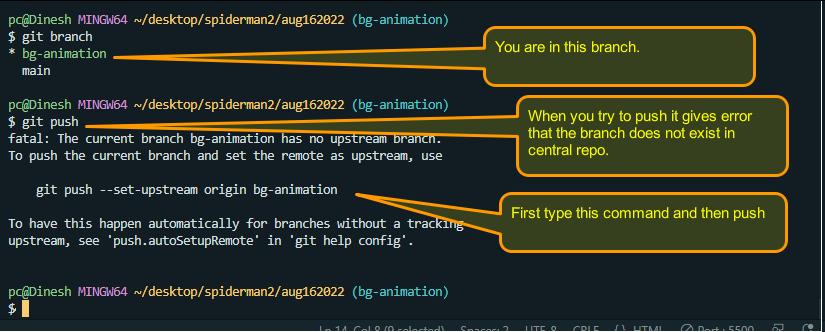
Git commands:

1. Git init - when you first initialize the git, it makes an empty repo. And .git folder is created automatically



1. ls -la to see hidden files
2. 
3. Hash value is the combination of foldername + filename. You can get the hash value from variable but you cannot get variable from hash value. Hash value is generated based on the contents of the file. 

*It will be asked if your 5+ years experience.*

1. When you give a password for ex dinesh@007 it will save this as a hash value. This hash value is same whenever you search from any part of the world.
2. Git uses SHA1 160bits.
3. There are 4 types of objects
4. Blob
5. Tree
6. Commits
7. Tag
8. 
9. All the versions are stored in repo. You can always go to any version and start working. You can also see what changes you have made in which version.
10. Now, these version control system has two types. 1) Centralized and 2) Distributed
11. Centralized :Here, every developer needs to connect the central repository. But the problem is if any one of the developers has connectivity issues, and if he wants to see the changes and progress made in the project, then he can’t access as he has no connection. This is the reason this centralized version control system has not been in use.
12. Distributed : Every developer has their own version control system and these are connected to central vc. If one developer connection has crashed, then he can easily access the other developer’s vc. They can make any other developer’s vc as centralized
13. Git-Hub and Mercurial are current VCs. 99% only Git-Hub is used.
14. 
15. Code . Opens the VS Code with the repository
16. We can directly open VS code and open the folder from the local machine.
17. These are the default folders created in the .git folder. This is a hidden folder created when a repo is made
18. 
19. You can work in git or on VScode. Vscode Terminal and git acts same
20. You can edit in VScode in Terminal or directly open the file in vscode. It automatically updates.
21. 
22. Git status To check if there are any untracked files or any commits
23. When a file is modified and a new file added, then it shows file is modified and untracked file for new file
24. git add filename adds to the staging area
25. git add . adds all files including modified and untracked files
26. git reset filename resets the file
27. git reset resets all files and brings back from staging area
28. Example below for staging area and untracked files
29. When you get an error when committing, for the first time you need to tell who you are. So you need to configure your email and name
30. git config - - global user.email “exampl.com”
31. git config - - global user.name “Your name”
32. git config - - user.email “example.com” if you don’t want it to know globally
33. git add . or specific filename adds to the staging area
34. git commit -m “message” adds to the local repository not to the *Git-Hub*
35. git log to see the status
36. Every commit gets the unique identifier and can be seen in git log
37. git pull shows the status of central repository. You need to check before pushing your commits to the central repository, because somebody else also must be updating the files.
38. git push to push the the local commits to the central repository where your modifications and new files will be reflected. Your local repository and git-hub repository will be in sync.
39. git checkout to go to any previous commit. (Copy the unique identifier from git log)
40. git checkout main to go to the last commit (*not going backwards. This moves forward*). This goes to the top of the commit (to the original state)
41. For suppose if you have created a file directly in git hub and committed there, you won’t see in your local repository. But if you want to update your local repository with central repository. Then the command is **git pull**. With **git pull** your local repository and central repository are in sync.
42. You can delete the file which was created in central repository in your local repository and push the changes so that it will be deleted in central repository also.
43. git branch shows which branch you are in
44. git branch example creates an example branch (This is only in your local repository). You need to push this branch if you want to see this branch in central repository which is git-hub
45. git checkout example you will be in example branch
46. You can use VScode to add and commit to a local repository. It is same like commands
47. When you made a few changes in your branch and now you would like to merge it with the main branch. Before merging, you need to check the difference between the main branch and your branch.
48. git diff your branch
49. git merge branch name to your current branch you are in. If you are in the main branch and using this command, then the specified branch will be merged to the main branch.
50. Directly merging the branch to the main branch is not the correct way. We need to show our changes first to our team leader to go through our changes and approve it. For that we use *pull request.*
51. But these pull requests happen at the Central repository level. So how do you make pull request when you are branch is not there yet. Your branch is still in local repository. For that you need to be in your branch and type the command git push - - set - upstream origin branchname.
52. git push - - set - upstream origin branchname pushes your locally made branch to central repo
53. git branch -d branchname to delete the branch
54. **git merge main this merges the main branch content to your branch (local, NOT central)**
55. Now for example h1 text in main is different than in your branch h1 text, and when you are trying to merge main branch content, then it says with error that it has a conflict and need to fix.
56. When you made some changes in a file and if check the status it shows file modified. Now if you think you don’t like the changes you can simply type *git checkout filename* The changes will be erased. This is okay and cool.
57. But if i have committed some changes and want to revert back, how? git reset HEAD~1 This means that it goes back to one commit from where you are at currently. And then type git checkout filename git reset HEAD~2 for 2 commits back
58. git reset <unique identifier> takes you to that state but it won’t erase
59. git reset - - hard <unique identifier> takes you to that state and also erases.

**Things to remember**

1. Create a branch
2. Switch to the new branch
3. Do the changes
4. Commit the changes
5. Merge the main to the current branch
6. Fix if there are any conflicts
7. Push the changes
8. Create the pull request.