**Git Commands**

1. **git status**

- Tells us the status of the git. It will show “fatal” if the git is not initialized in the current directory, otherwise it will tell us other details.

2. **git init**

- This will initialize the current directory as a git repository.

3. **git add --a**

- This will add all files to the staging area, making them ready for commit.

4. **git add fileName**

- This will add a specific file to the staging area.

5. **git commit -m “Commit Message”**

- This will commit all the files in the staging area to the git repository and the “Commit Message” will be stored and displayed to let others know, what did this commit achieved.

6. **git log**

- This will generate a log displaying all the commits along with their respective times and messages.

7. **git clone URL**

- This will clone an online repository into the current directory. We can also change the name of the online repository by adding a space after URL and adding new name as

**git clone URL newName**

8. To ignore some files to track in git repository

a.) Create text file named ".gitignore" using command (if not exist):

**touch .gitignore**

b.) Add the name of the file to be ignored in .gitignore file

9. To ignore every file on the basis of its extension, write \*.extension in the .gitignore file.

Example : if we want to ignore all files having .log extension, open .gitignore file and add \*.log in that file.

10. **git diff**

It compares working directory with the staging area.

11. **git diff --staged**

It compares staging area with the last commit made.

12. **git commit -a -m “Message”**

This command will take all the modified files and commit them directly, while skipping the staging area. It can only commit changes made in already existing files, but it cannot commit any untracked files (newly created files).

13. **git rm fileName**

It will remove the given file from the working directory and stage the changes to be committed.

14. **git mv currentName newName**

This will rename the file in working directory having currentName to newName and stage the changes to be committed.

15. **git rm --cached fileName**

This command will remove the file from being tracked by the git. This is useful, when we want to ignore a file and put it in .gitignore at later stages. Since, the file is already being tracked by the git, we need to explicitly untrack it because otherwise the git would not ignore it even if it is placed in .gitignore.

16. **git log -p**

This command will show us all the commits along with the differences in before and after the commit, i.e. it will describe all the modifications made in the commit.

17. **git log -p -n** Here n is any integer number

This command will only show us the data of ‘n’ latest commits.

18. **git log --stat**

This command will show us the brief stats of modifications done in the commit. We can see how many changes (both insertions and deletions) have been made in each file.

19. **Git log formatting:-**

a] **git log --pretty=oneline** -> Shows commit history in one-liners.

b] **git log --pretty=short** -> Shows commit history in short form.

c] **git log --pretty=medium** -> Shows commit history in medium form.

d] **git log --pretty=full** -> Shows commit history in longer form.

e] **git log --pretty=fuller** -> Shows commit history in extended form.

e] **git log --pretty=format:”specified\_format”**

With this command we can specify the format in which we want to see the commit log. The format has its own unique characters which can be seen online.

20. **git commit --amend**

With this command we can amend the last made commit with the files currently present in staging area. This is useful, when sometimes we make a commit but then realise, we left some modifications, and it seems absurd to make a new commit for some minor changes.

In such a case, we can just add the changed files into staging area and amend them to the last made commit.

By default, if we have VIM editor, we need to first press ‘I’, which will take us to insert mode. There we can make necessary changes. Afterwards, we need to press ‘Esc’ key, to exit insert mode. Then we press ‘:’ and ‘wq’ to finalise the changes.

If you have VS code editor, it is pretty simple, you just change the commit message, save and close the file and it will do the rest for you.

21. **git restore --staged fileName**

This command will unstage the specified file from staging area. That is their status will change from staged to being unmodified.

22. **git checkout -- fileName**

This command will restore the file version from the last made commit. This is helpful, because sometimes, we mess up the file and want to revert back, when it was working fine.

23. **git checkout -f**

This command will restore whole directory to the last made commit. That is all the modifications done will be cancelled and we will get the working directory in the stage corresponding to last made commit.

24. **git remote add origin /path/to/your/online/repository**

This command will enable your device to be attached to an online repository, and the repository is named ‘origin’ by default.

25. **git branch -M main**

With this command, you can push your locally existing git repository to the main branch of the remote repository.

26. **git push -u origin main**

This command will push all the changes and commits made on your local repository to the remote repository.

27. **git config --global alias.<aliasName> “real command”**

With this command, we can set up aliases for different commands in git. This will enable us to simply call the alias and the corresponding command will run automatically. This is useful, as instead of typing longer commands we can just type shorter aliases.

e.g. If we are constantly checking the last commit, we can setup an alias ‘last’ such as:

git config --global alias.last ‘log -p -1’

And then, whenever we call git last, we will be presented with data of last made commit.

28. **git checkout -b <branchName>**

This command will create a new branch having specified name.

29. **git checkout <existingBranch>**

This command will switch the current directory to the specified branch and all the files will transition accordingly.

30. **git branch**

This command will list all the branches present in your repository.

31. For merging branches, switch to parent branch, and run

**git merge <branchName>**

32. **git push <remoteRepoName> <branchName>**

This command will push the specified branch from local repository into the given remote repository.

33. **git push <remoteRepoName> --all**

This command will push all branches of the local repository into the given remote repository.

34. **git push -d <remoteRepoName> <branchName>**

This command will delete the specified branch from the remote repository.

**Linux Commands**

1. **pwd** (present working directory)

- Tells us about the directory we are currently in.

2. **cd ‘/path/to/directory’**

- This will change our working directory

3. **ls**

- This will list all the files in the current directory.

4. **rm -rf ‘filename’**

- This will remove the given file from current directory.

5. **touch ‘filename’**

- This will create a file with given name (if already not exists) in the current directory.