Git command :

New or existing folder or code to push into github process:

1. Which project want to push into git , mount directory to the same project

Eg. If I want to push project “Microservice” then I will mount my directoyr to Microservice.

Inthis case only contenct inside microservices project or folder will pushed to git hub

1. If we want to push “Microservice” folder in github -> then we need to mount the directoy to its parent or precedent , suppose: Microservice is under documents then we need to mount directory to Documents and should follow the following process

echo "# masterLearning" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Shambhu-repo/masterLearning.git

git push -u origin main

|  |  |  |
| --- | --- | --- |
| S>N | Command : | Description |
| 1 | git init | Initializing a repository |
|  | rm -rf .git | Un do initialization |
|  | Staging files |  |
| 1 | Git add <filename> | Stages a file |
|  | Git add <file name1> <file name2> | Stages multiple files |
|  | Git add \*.js | Stage with pattern of js |
|  | Git add . | Stages the current directory with all content |
|  |  |  |
|  | Viewing the status |  |
|  | Git status | Full status provides |
|  | Git status -s | Short status |
|  |  |  |
|  | Committing the staged files |  |
|  | Git commit -m “msg” -m “msg” | Commits with two line message |
|  | Git commit | Opens the default editor to type a long msg |
|  |  |  |
|  | Skipping the stagging area |  |
|  | Git commit -am “message” |  |
|  |  |  |
|  | Viewing the staed/unstaged changes |  |
|  | Git diff | Shows unstaged changes |
|  | Git diff --staged | Shows staged changes |
|  | Git diff --cached | Same as above |
|  |  |  |
|  | Viewing the history |  |
|  | Git log | Full history |
|  | Git log --oneline | summary |
|  | Git log --reverse | Lists the commits from the oldest to new |
|  |  |  |
|  | Viewing a commit |  |
|  | Git show 921a2ff | Shows the given commit |
|  | Git show HEAD | Shows the last commit |
|  | Git show HEAD~2 | Two steps before the last commit |
|  | Git show HEAD:file.js | Shows the version of file.js stored in the |
|  |  |  |
|  | Unstaging files (undoing git add) |  |
|  | Git restore –staged <files name> | Copies the last version of file.js from repo |
|  |  |  |
|  | Discarding local changes |  |
|  | Git restore <filename> | Copies files from index to working derectory |
|  | Git restore<filename1> <filename2> | Copies multiple file in working dirctory |
|  |  |  |
|  | Viewing the history |  |
|  | Git log --stat | Shows the list of modifies files |
|  | Git log --patch | Shows the actual changes(patches) |
|  |  |  |
|  | Filtering the history |  |
|  | Git log -3 | Shows the last 3 entries |
|  | Git log --author = “Mosh” |  |
|  | Git log –before= “2020-08-17” |  |
|  | Git log –after =” one week ago” |  |

|  |  |  |
| --- | --- | --- |
|  | Git log –grep = “GUI” | Commits with GUI in their message |
|  | Git log -s “GUI” | Commits with GUI in their patches |
|  | Git log hash1….hash3 | Range of commits |
|  | Git log file.txt | Commit that touched file.txt |
|  |  |  |
|  | Formatting the log output |  |
|  | Git log –pretty=format: “%an committed%H” |  |
|  |  |  |
|  | Creating an alias |  |
|  | Git config –global alias.lg “ log –online” |  |
|  |  |  |
|  | Finding a bad commit |  |
|  | Git bisect start |  |
|  | Git bisect bad | Marks the current commit as a bad commit |
|  | Git bisect good ca49180 | Marks given commit as a good commit |
|  | Git bisect reset | Terminate the bisect session |
|  |  |  |
|  | Finding shortlog |  |
|  | Git shortlog |  |
|  |  |  |
|  | Viewing the history of a file |  |
|  | Git log file.txt | Shows the commits that touched file.txt |
|  | Git log –stat file.txt | Shows statistics(the number of changes) |
|  | Git log –patch file.txt | Shows the patches (changes) applied to file.tx |
|  |  |  |
|  | Finding the author of lines |  |
|  | Git blame file.txt | Shows the author of each line in file.txt |
|  |  |  |
|  | Branching and merging |  |
|  | Git branch bugfix | Creates a new branch , bugfix |
|  | Git checkout bugfix | Switches to bugfix |
|  | Git checkout -C bugfix1 | Create and checkout bugfix1 |
|  | Git branch -d bugfix | Delete the bugfix branch |
|  |  |  |
|  | Comparing branches |  |
|  | Git log master bugfix | Lists the commits in the bugfix and master |
|  | Git diff master bugfix | Shows the summaty of changes |
|  |  |  |
|  | Stashing |  |
|  | Git stash push -m “new tax rules” | Create new stash |
|  | Git stash list | List all the stashes |
|  | Git stash show stash@{1} | Shows the given stash |
|  | Git stash show1 | Short cut for stash@{1} |
|  | Git stash apply1 | Applies the given stash |
|  | Git stash drop1 | Delete the given stash |
|  | Git stash clear | Delete all the stashes |
|  |  |  |
|  | Viewing the merged branches |  |
|  | Git branch --merged | Shows the merged branch |
|  | Git branch –no-merged | Shows the unmerged branches |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  | Rebasing |  |
|  | Git rebase master | Changes the base of the current branch |
|  |  |  |
|  | Cherry picking |  |
|  | Git cherry-pick dad47ed | Applies the given commit on the |
|  |  |  |
|  | Collaboration |  |
|  | Git clone <url from git repo> | Cloning a repository |
|  |  |  |
|  | Syncing with remotes |  |
|  | Git fetch origin master | Fetches master from origin |
|  | Git fetch origin | Fetches all object from origin |
|  | Git fetch | Short cut for “git fetch origin” |
|  | Git pull | Fetch+merge |
|  | Git push origin master | Pushes master to origin |
|  | Git push | Short cut for push origin master |
|  |  |  |
|  | Sharing tags |  |
|  | Git push origin v1.0 | Pushes tag v1.0 to origin |
|  | Git push origin -delete v1.0 |  |
|  |  |  |
|  | Sharing branches |  |
|  | Git branch -r | Shows remote tracking branches |
|  | Git branch -vv | Show local and remote tracking branch |
|  | Git push -u origin bugfix | Pushes bugfix to origin |
|  | Git push -d origin bugfix | Removes bugfix from origin |
|  |  |  |
|  | Rewriting History |  |
|  | Git reset –soft HEAD^ | Removes the last commit, keeps change |
|  | Git reset –mixed HEAD^ | Unstages the changes as well |
|  | Git reset –hard HEAD^ | Discard local changes |
|  |  |  |
|  | Reverting commits |  |
|  | Git revert 72856ea | Reversts the given commit |
|  | Git revert HEAD~3 | Reverse the last three commits |
|  | Git revert –no-commit HEAD~3 |  |
|  |  |  |
|  | Recovering lost commit |  |
|  | Git reflog | Shows the history of head |
|  | Git reflog show bugfix | Shows the history of bugfix pointer |
|  |  |  |
|  | Amending the last commit |  |
|  | Git commit --amend |  |
|  |  |  |
|  | Interactive rebasing |  |
|  | Git rebase -I HEAD~5 |  |

Note : Git revert <commitID> ( this command will revert back the code in local , meaning it will get previous stated of code in local . So we need to again push it to repository to get previous state of code . If some one ask me to revert back code so I do given command and again I will push it to repo so I will have reverted code in repo )