git config --global user.email ‘deepesh123123@gmail.com’

git config--global user.name ‘deepesh87’

git init -- command creates a new Git repository. It transforms the current directory into a Git repository.

git add filename -- this will add the file to the staging area to be commited. The tracking of the file will start only once this is run

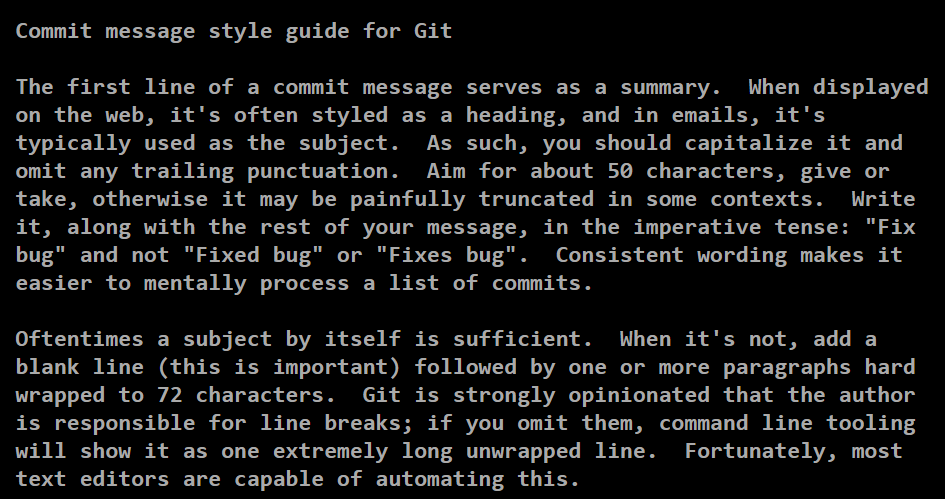
git status

git commit -- this will open up the editor to write the commit message

git commit -m ‘commit message here’ – to directly write the commit message

git log -- to view all the commits done so far

git config -l – list details like the username, name etc



git commit -a -m “message” 🡪 is a shortcut to stage any changes to tracked files and commit them in one step. So, for all files getting tracked (i.e. on which the git add command has been run earlier) can be directly pushed to commit by running this. i.e. git add filename + git commit -m ‘message’ in 1 step

git log -- to view all the commits done so far

git log -p 🡪to view all the commits done so far and also see come code changes that were made (p for patch as we can see the changes just like we can in patch file)

git show (git show commit ID) will show only the last/particular commit in details

git log –stat 🡪 shows some stats from each commit, like how many files were changed, how many line added/inserted/deleted etc

git diff (git diff filename) 🡪 lets us see all the changes we have made since the last commit.

git add -p 🡪 to review changes before adding them to the staging area

echo filename > .gitignore (or echo filename >> .gitignore) to ignore the file from being tracked

THEN git add .gitignore🡪 THEN git commit -m ‘message’

|  |
| --- |
| ################### |
|  | \*.com |
|  | \*.class |
|  | \*.dll |
|  | \*.exe |
|  | \*.o |
|  | \*.so |
|  |  |
|  | # Packages # |
|  | ############ |
|  | # it's better to unpack these files and commit the raw source |
|  | # git has its own built in compression methods |
|  | \*.7z |
|  | \*.dmg |
|  | \*.gz |
|  | \*.iso |
|  | \*.jar |
|  | \*.rar |
|  | \*.tar |
|  | \*.zip |
|  |  |
|  | # Logs and databases # |
|  | ###################### |
|  | \*.log |
|  | \*.sql |
|  | \*.sqlite |
|  |  |
|  | # OS generated files # |
|  | ###################### |
|  | .DS\_Store |
|  | .DS\_Store? |
|  | .\_\* |
|  | .Spotlight-V100 |
|  | .Trashes |
|  | ehthumbs.db |
|  | Thumbs.db |

**To delete a file from staging area:**

Imagine a file is already added. So, it is on the staging area. We now decide to delete the file.

git rm filename 🡪 git commit -m ‘message’

**To rename a File:**

git mv old\_name new\_name

git commit -m ‘message’

**To reverse a change before Committing:**

1. **before Staging the changes:**

**git checkout filename** [ note the file should have been tracked i.e. file should have added to staging area at some point in the past]

This will undo the changes that is made to the code and the changes will be lost

1. **after Staging the changes:**

git reset HEAD filename [ Note: This will undo the staging that is done, this won’t delete the changes that has been done]

**TO EDIT THE LAST COMMIT:**

git commit –amend 🡪 overwrite the previous commit

git revert HEAD -🡪 a new commit is created with inverse changes. This cancels previous changes instead of making it as though the original commit never happened.

**TO GO BACK TO PREVIOUS COMMIT:**

Git revert HEAD 🡪 goes to the previous commit.

Git revert COMMIT ID 🡪 goes to the particular commit

**BRANCHES:**

git branch 🡪 view all branches, current branch will be in Green with a \* pointed to it

git branch any\_name 🡪 creates this new branch but will NOT point to it yet

git checkout any\_name 🡪 switches to any\_name branch from current branch

*Once we move into any NON master branch, any new files that we create in the new branch wont be seen (when we do ls -l) in the master branch.*

*Git log on the master branch won’t show the commits made on the non-master branches.*

*Note 1: If we create a new file while in master and do not track it yet ( do not run git add file\_name) and checkout to another branch, the same file will then be available to be added (tracked) on the other branch. [ of course, all files in master are anyways available in the non-master branch ]*

*Note 2: If we create a new file while in non-master and do not track it yet (note I haven’t run git add filename), If I then checkout to master branch, the same file will be seen while in the master branch aswell.*

*Note 3: I created a file in non-master, added it in non-master ( by running git add file), then checked out to master and could see the same file as if it was added from master. I was able to commit the same file from master. Went back to check it on non-master, could not see it, so commit on master closed the deal.*

*git checkout -b new\_branch 🡪 to create and switch to the new branch directly*

*git branch -d new\_branch 🡪 to delete a local branch. ( if any file remains left which is not added then delete won’t work, use D)*

*git push origin --delete branch\_name 🡪 to delete a remote branch*

*Git branch -D new\_branch 🡪 force delete a branch*

*git merge new\_branch\_name 🡪 This will merge the changes made in the new branch into the master branch. Git log from the master branch will now also show the commit made in the new branch. Always stay at the master branch to merge any other branch into it.*

*git merge –abort 🡪 If there are merge conflicts (meaning files are incompatible), --abort can be used to abort the merge action.*

*git log –graph--oneline 🡪 To see 1 line per commit and the merge as a graph*

# Distributed VCS

Git is a Distributed VCS. Each Developer has a copy of the whole repository on their local system.

Git clone (link of a repository from github) 🡪 this will copy all the files from the remote repository to local. A directory with the name of the repo is automatically created on local and will have the working tree in it.

Git push 🡪 to push commits from the local repository to the remote

Git config --credential.helper cache 🡪 To enable the credential to be cached so we don’t have to enter it multiple times (they are cached for 15 minutes) (https://docs.github.com/en/github/getting-started-with-github/caching-your-github-credentials-in-git)

# REMOTE repository:

git remote show origin 🡪 Whenever some other user has made changes to the master branch (in the remote repository), we can run this command to see if our local repo has the latest or not. This will tell us that the local is out of date/local upto date.

Git fetch 🡪 This command copies the commits done in the remote repository to the remote branches

Git log 🡪 will show the changes in the local repo.

Git log origin/main 🡪 this will show changes in the remote repo and main branch (this used to be Git log origin/master earlier was renamed by git)

Git status at this point will tell us that there is a commit on remote repo that we don’t have in local

Git merge origin/main 🡪 to merge the changes of the main branch of the remote repo into our local repo

***git fetch fetches remote updates but doesn't merge, we merge them manually by using git merge origin/main; git pull fetches remote updates and merges.***

Git fetch will only merge in the local repo if the same branch is also present in repo ( for example in remote a change was made in branch main, then git pull will add those changes to the local repo main branch).

**Case: But if we only have 1 branch say master in local repo and on the remote repo changes were made on a new branch after creating it, git pull will not automatically merge them into local.**

First run git remote show origin. This will tell us that a new branch is created on remote apart from the origin main branch. This will however tell us that the current branch which is main is upto DATE.

We will have to call git pull and then create the other branch using git checkout branch\_name. Then commits of the remote non main branch will automatically come to the other branch in local.

<https://git-scm.com/docs/git-remote>

# MERGE CONFLICT:

This happens for example when I make a change in a file and the file is changed by someone else on the remote branch. Git will try its best to resolve and merge the changes but if the changes done by both are for example in the same line, Git will need a manual conflict resolution. This is how we start.

First try to push your local changes into the remote using –

Git add filename 🡪 git commit -m ‘some comment’ 🡪 git push. Git will reject your change, and this should throw an error with details of the conflict. This will also tell that there are changes that we do not have in our local branch. So we will have to pull the changes that we donot have in local by using 🡪 git pull

Git will now try to merge the changes but may show a conflict. We will have to resolve this manually. 🡪 git log –graph –oneline –all will show us the conflict in details.

* Git log -p origin/main to see the changes made in remote.

Now, we can open the file and look for <<< and resolve the conflict ourselves by deleting what we donot need. After doing this, add the changes and commit once again.

Git add filename

Git commit -m ‘comment’

And then finally git push

# PUSHING TO REMOTE NON-MAIN BRANCH (CREATE A NON-MAIN BRANCH FROM LOCAL TO REMOTE):

Create a new branch using -🡪 git branch -b branch\_name. this will create this new branch and switch to it. Then make the changes, add the changes to staging area, commit the changes. Then While pushing the changes use the following as we will be pushing the changes to a new non main branch on remote repository.

🡪 Git push -u origin branch\_name. this will create the branch and add the changes made in local to the remote non main branch. [ note: Give the same name as in the local non main branch]

GIT REBASE MASTER 🡪 THIS WILL REBASE THE CURRENT BRANCH ON THE MASTER BRANCH.

Forking is a way of creating a copy of the given repository so that it belongs to our User.

Pull Request is a commit or series of commits that you send to the owner of the repository so that they incorporate it into their tree.

SQUASH changes into a Single Commit before creating a PULL REQUEST:

Git rebase -i master 🡪 This will open a text editor with all the commits made until now, from the oldest to the most recent. By changing the first word of each of the lines, we can decide what we want to do with the commits. Default action is pick which takes the commits and rebases it against the branch we selected.

We can use any of the options given in comments. In Both squash and fixup the contents of the commits are merged into the previous commit in the list.

Squash🡪 commit messages are added together and new editor opens to make any changes  
Fixup 🡪 commit message for that commit is discarded.

**Continuous Integration System (CI):** will build and test our code every time there is a change (new commit or PR in the main branch).

Once we have our code automatically built and tested, the next automation step is-

**Continuous Deployment/ Continuous Delivery (CD):** means new code is deployed often to avoid rollouts with lots of changes between 2 versions to avoid errors.

Jenkins is one such platform that is used for CI/CD. Travis can also be used which integrates with Github.

**Artifacts:** The name used to describe any files that are generated as part of the pipeline.