* **git checkout -b <branch\_name>:** will create a new branch and checks out directly into it.
* **git commit -am “some\_message”:** used to directly add and commit the changes.
* **git reset:** used to undo the commits.
* **git diff:** a powerful tool for reviewing changes in your code, allowing you to see what has been modified, added, or removed before making commits or merges.
* **git diff main origin/main:** used to show the differences between your local main branch and the main branch on the remote repository.
* **git rebase —continue:** if there are any conflicts during rebasing, we need to clear the conflicts, after clearing the conflicts, we use this command to resume the rebasing.
* **git log —graph —all:** used to display all the commits with details like author, commit message and commit time.
* **git log —graph —oneline —all:** used to display all the commit history, each commit details is displayed only in one line.
* **echo some\_text> file\_name:** used to enter the data into the given file. Even if some data is present in the file, the data will be replaced by the data given in the command.
* **git reset —soft HEAD~1:** used to undo the no. of commits given in the command, **soft** is used to undo the commit to staged file state.
* **git reset —mixed HEAD~1:** used to undo the no. of commits given in the command, **mixed** is used to undo the commit to unstaged file state.
* **git reset —hard HEAD~1:** used to undo the no. of commits given in the command, **hard** is used to discard the commits and also the changes associated with them. This will bring the repository back to it’s earlier state.
* **git stash:** used to temporarily save the changes in a Git repository without committing them.
* **git stash pop:** used to remove the most recent file from the stash list.
* **git log -p -n:** gives the information of the latest n no. of commits.

**MERGE CONFLICT:**

⇒ When two developers work on the same file in two different branches and want to merge both the branches then there will be a conflict.

⇒ Because there are changes in both the branches for the same file. The data in both the files will be merged into a single file during merging and an error will be generated.

⇒ Now we need to open the file with the conflict manually and edit it manually.

⇒ When two files are merged into a single file, the file starts with “<<<<<<< HEAD” and the data in both the files are divided by “=======” as the center line and ends with “>>>>>>> branch\_being\_merged.”

**REBASE:**

⇒ The main aim of rebasing is to maintain a straight and cleaner commit history. This makes it easier to navigate the commits.