**GIT Commands: -**

1. **Git init**: - It will create empty git repository in the form of .git folder/directory.
2. **Git status**: - To know the state of the file.

* If files are in working directory(WD) it’ll show nothing to commit,WD is clean.
* If file is in the staging area, but not commited, it shows with git status.

1. **Git add <filename>**: - To add particular file to the staging area for git.
2. **Git add .** : - To add all file to the staging area for git.
3. **Git commit**: - To record the changes made to the files(staging area) in a local repository.
4. **Git config**: - To assign user name and user email to commits from a local computer.

Git config user.name “Radhika”

Git config user.email “[radhika123@gmail.com](mailto:radhika123@gmail.com)”

**(OR)**

Git config - -global user.name “Radhika”

Git config - -global user.email “[radhika123@gmail.com](mailto:radhika123@gmail.com)”

1. To UNSET configurations.

* Git config - -global - -unset user.name “radhika”
* Git config - -global - -unset user.email “[radhika123@gmail.com](mailto:radhika123@gmail.com)”

1. **Git branch**: - To list all the branches
2. **Git branch <branch\_name>**: - To create new branch without **checking** **out** from the **current** branch.
3. **Git checkout < stage (branch name)>**: - To switch/check-in into <stage> branch.
4. **Git merge <stage>**: - By staying in **master branch** we have to type this command. It will merge all the files from **stage** branch to **master** branch.
5. **Git branch -d <stage>**: - To **delete** branch.
6. **Git checkout -b <dev>**: - To **create** a branch and **checkin** simultaneously. (or)

**git switch -c new-branch** or **git switch - -create new-branch**

1. **Git remote add origin <GR url>**: - To connect our local repository with a global/Remote repository.
2. **Git clone**: - To create a local working copy of/to download an existing remote repository. (or) use to create local copy of a remote repository.

Git clone <GR url>

1. **Git log**: - To know the commit history.
2. **Git reflog**: - To know complete history of commits.
3. **Git log - -oneline** :- To know commit with message in a single lines.
4. **Git log -p**: - To know complete details of each commits.
5. **Git revert**: - To undo the changes made to the file.

* If you want to get back the previous version of the file after some commits then use

This command: - **git revert <commit id>**

**Note:** commit id of the commit which you want to revert.

* **Git revert HEAD: -** Itwill revert the last commit to the previous version.

21.**Git rebase**: - Use to move current work to our master branch.

Takes a set of commits, copies them, and stores them outside your repository.

22.**Git pull**: - To download the latest changes from remote repository to local repository. (or) used to update an existing local repository with the latest changes from the remote repository.

23. **Git reset**: - To remove changes from staging area and To undo commits at repository level. Git add <file.txt> 🡪 staging area

Git reset <file.txt> 🡪 unstaging area (working directory)

24.**Git chechout**: - git checkout is used to discard the changes in the working repository.

**Ex**: git commit <file name>

25. To undo particular commit :-

Git reset –hard Head~1 here,1 is commit number/commit id.

26.**Git restore** :- To unstage or even discard uncommited local changes.

Git restore - -staged <filename> 🡪To unstage the file from satging area before commit.

Git restore <filename> 🡪To discard local stages from working directory.

NOTE :-

* Before a file is available to commit to a repository, the file needs to be added to the Git index (Staging area).
* Each commit has a unique ID.
* **Git merge and Git rebase** are used to merge the work of different branches.
* **Git switch and git checkout** are used to switch between the branches.
* **Git reset** will not affect the working directory.

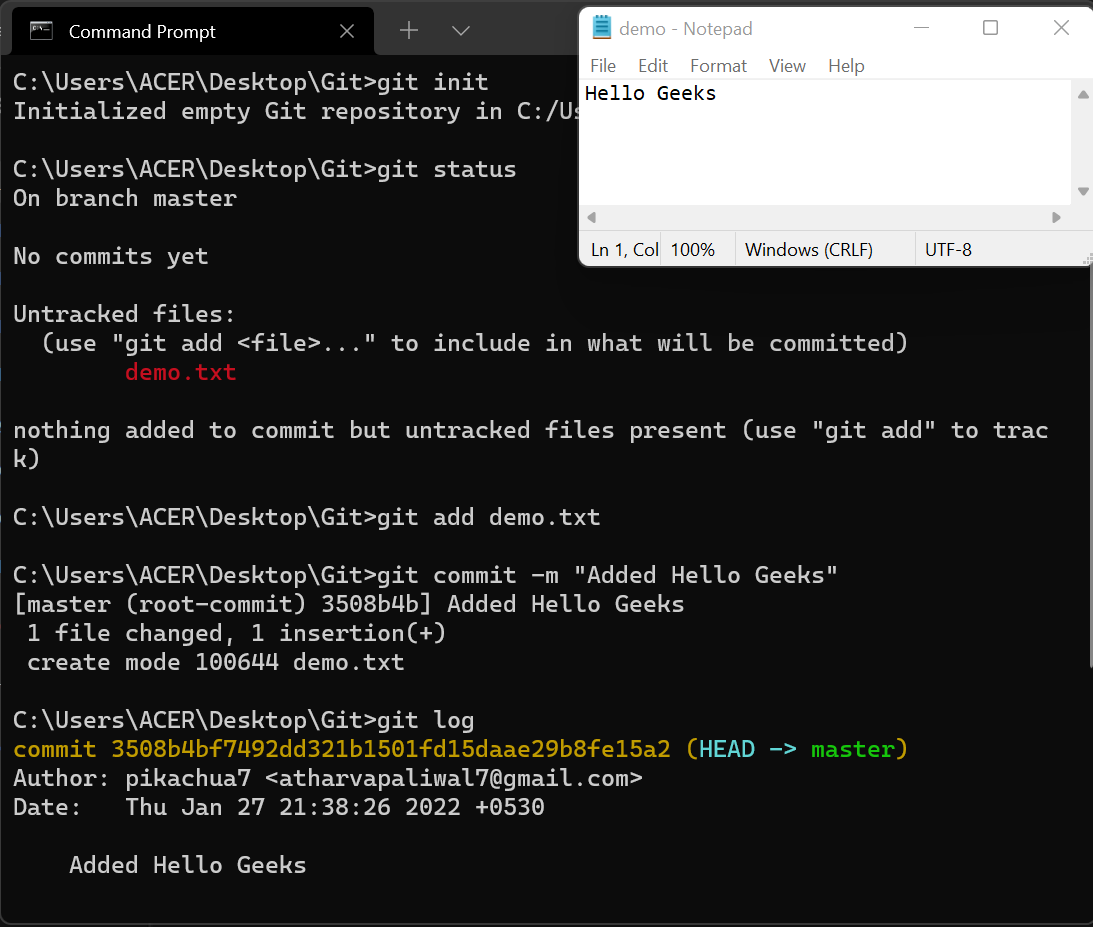
**Difference between git revert, git reset and git checkout.**

<https://www.geeksforgeeks.org/git-difference-between-git-revert-checkout-and-reset/>

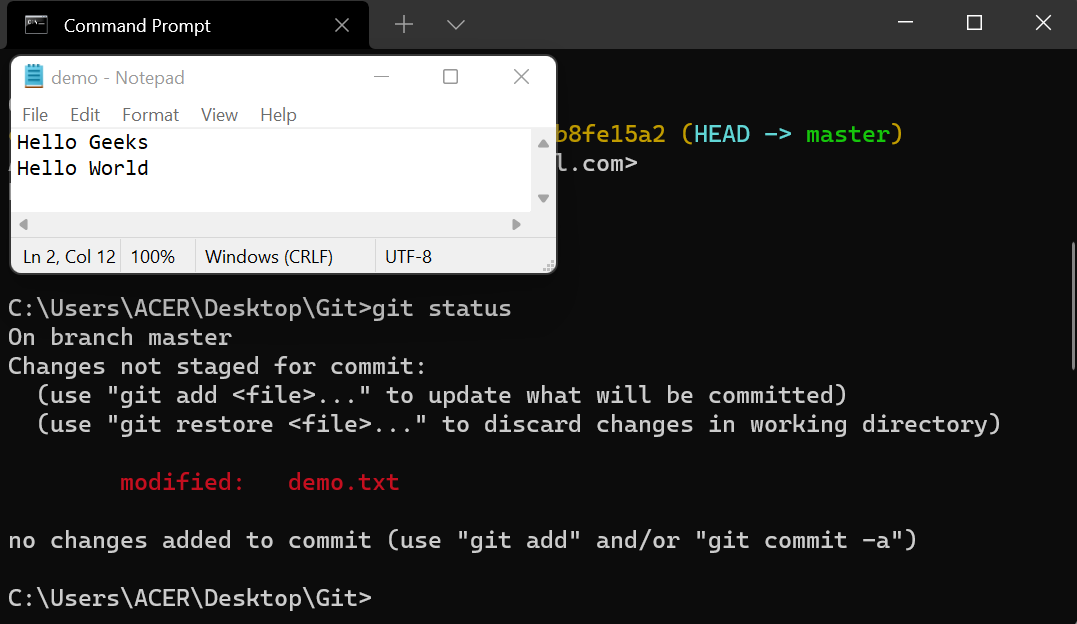
**Git checkout, Git reset, Git revert :-**

While Working with Git in certain situations we want to undo changes in the working area or index area, sometimes remove commits locally or remotely and we need to reverse those changes. There are 3 different ways in which we can undo the changes in our repository, these are ***git reset***, ***git checkout***, and ***git revert***. git checkout and git reset in fact can be used to manipulate commits or individual files. These commands can be confusing so it’s important to find out the difference between them and to know which command should be used at a particular point of time.

Let’s make a sample git repository with a file ***demo.txt***and “***Hello Geeks***” written inside it.



We can see that we have a single commit is done and the text document that has been committed with Hello Geeks in it. Now let’s add some more text to our text document. Let’s add another line ***Hello World.***Doing this change, our file now needs to be added to the staging area for getting the commit done. This updates are currently in the working area and to see them we will see those using ***git status***.

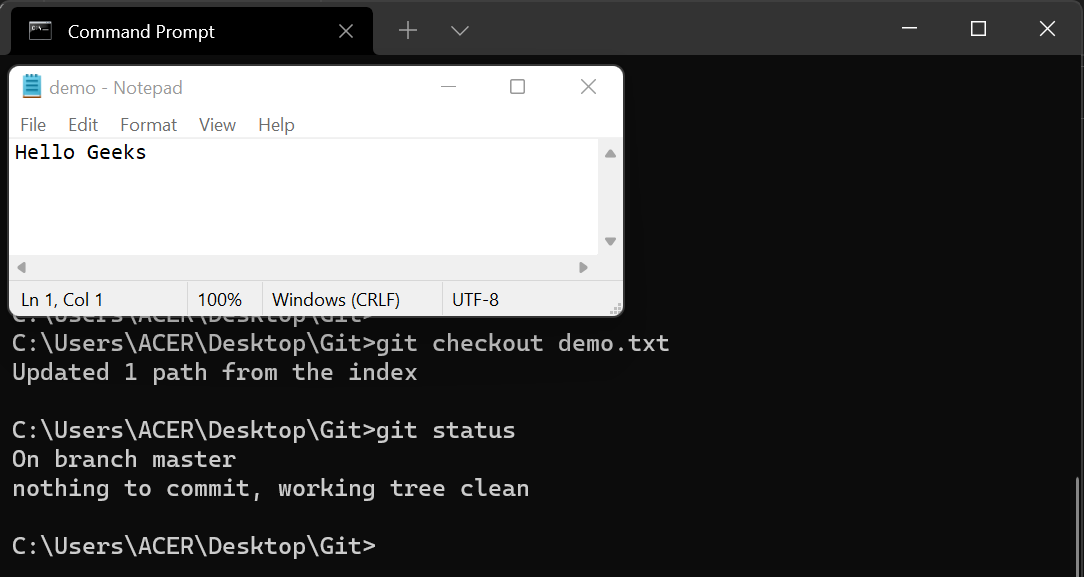


Now we have a change ***Hello World*** which is untracked in our working repository and we need to discard this change. So, the command that we should use here is –

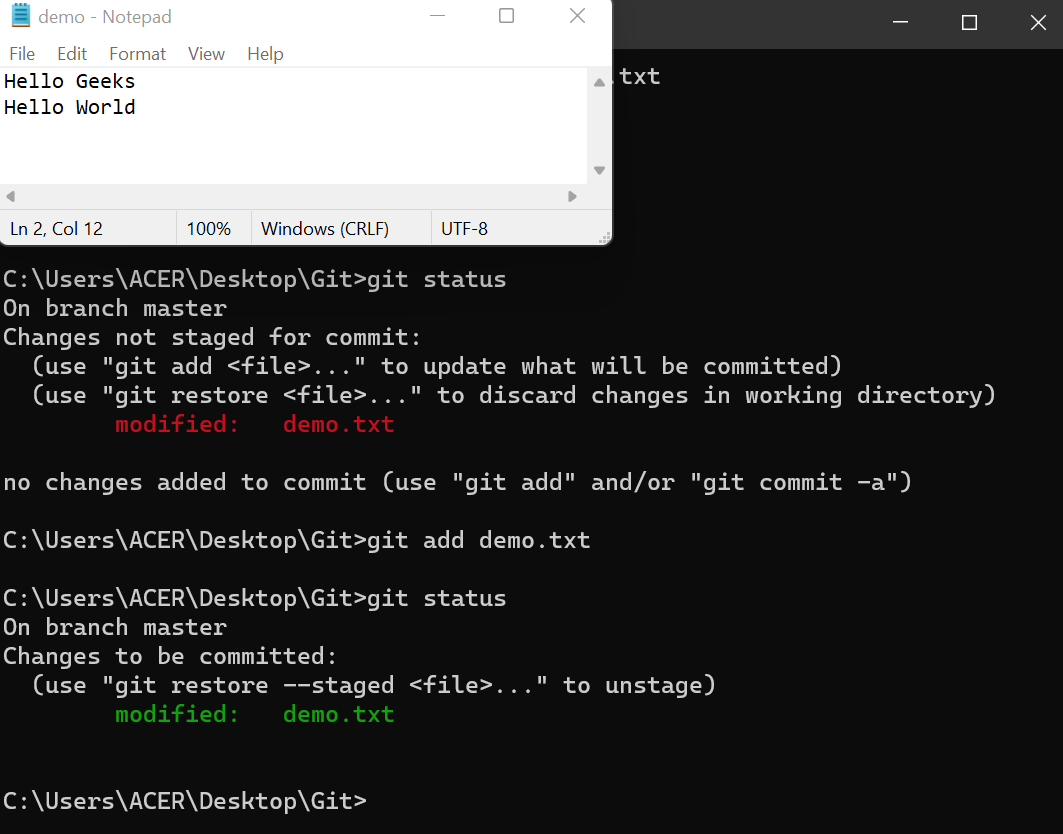
1. **git checkout**

git checkout is used to discard the changes in the working repository.

*git checkout <filename>*



When we write git checkout command and see the status of our git repository and also the text document we can see that our changes are being discarded from the working directory and we are again back to the test document that we had before. Now, what if we want to unstage a file. We stage our files before committing them and at a certain point, we might want to unstage a file. Let’s add ***Hello World*** again to our text document and stage them using the ***git add***command.



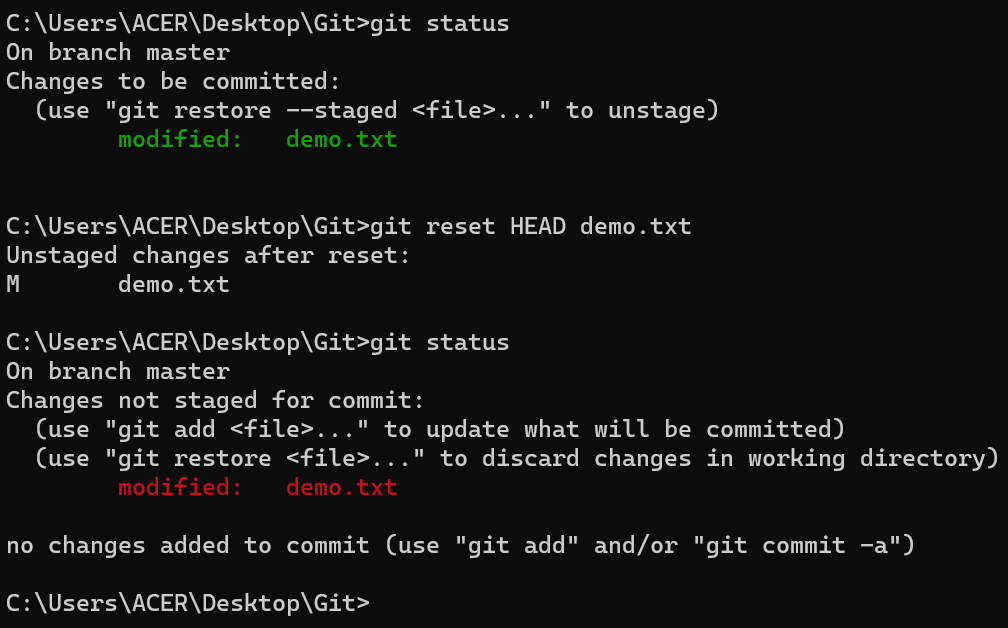
We want to unstage a file and the command that we would be using to unstage our file is –

1. **git reset**

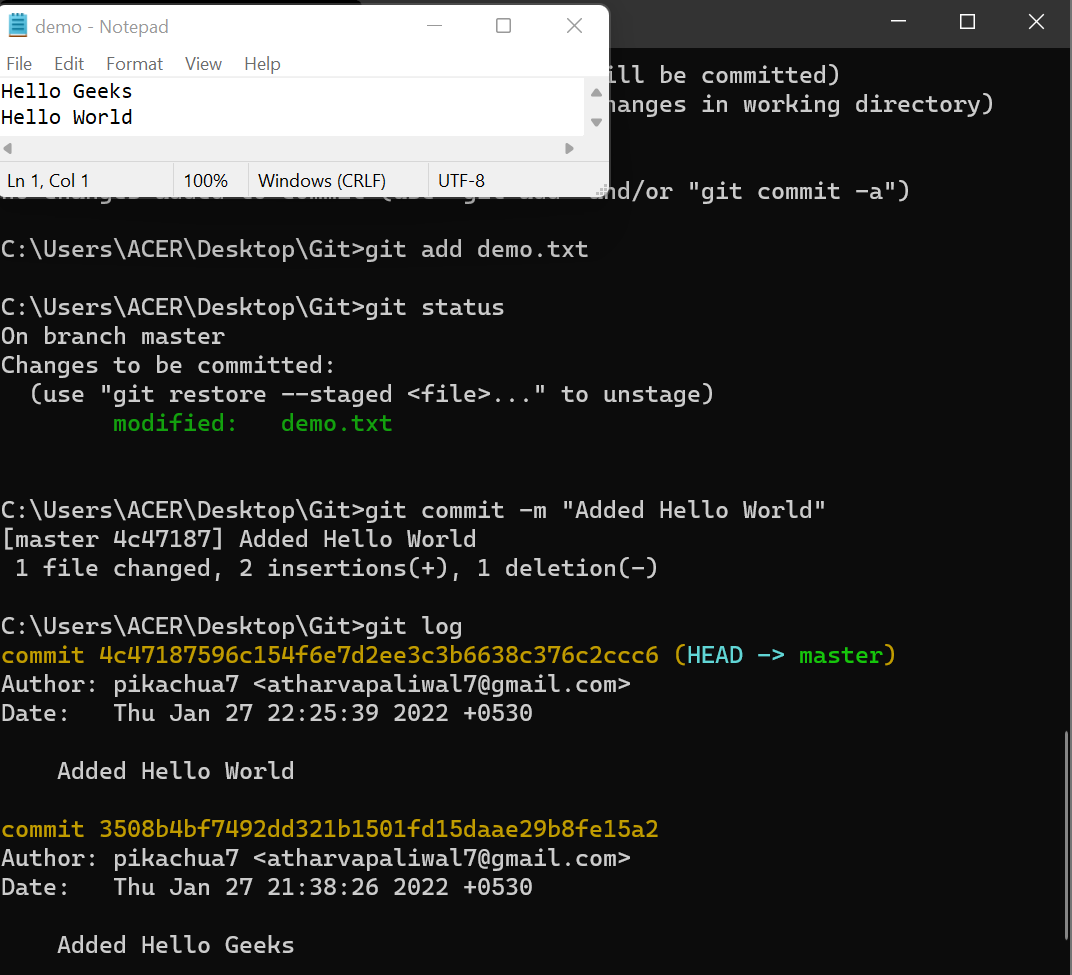
git reset is used when we want to unstage a file and bring our changes back to the working directory. git reset can also be used to remove commits from the local repository.

*git reset HEAD <filename>*

Whenever we unstage a file, all the changes are kept in the working area.



We are back to the working directory, where our changes are present but the file is now unstaged. Now there are also some commits that we don’t want to get committed and we want to remove them from our local repository. To see how to remove the commit from our local repository let’s stage and commit the changes that we just did and then remove that commit.

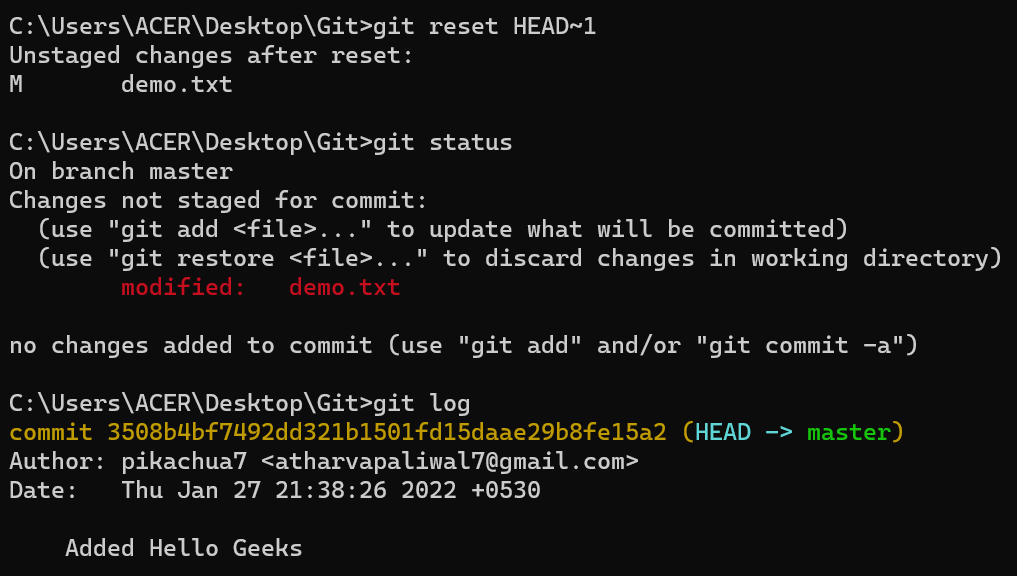


We have 2 commits now, with the latest being the Added Hello World commit which we are going to remove. The command that we would be using now is –

*git reset HEAD~1*

*Points to be noted –*

* *HEAD~1 here means that we are going to remove the topmost commit or the latest commit that we have done.*
* *We cannot remove a specific commit with the help of git reset , for ex : we cannot say that we want to remove the second commit or the third commit , we can only remove latest commit or latest 2 commits … latest N commits.(HEAD~n) [n here means n recent commits that needs to be deleted].*



After using the above command we can see that our commit is being deleted and also our file is again unstaged and is back to the working directory. There are different ways in which git reset can actually keep your changes.

* ***git reset –soft HEAD~1****–*This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.
* ***git reset –mixed HEAD~1*** or ***git reset HEAD~1 –***This is the default command that we have used in the above example which removes the commit as well as unstages the file and our changes are stored in the working directory.
* ***git reset –hard HEAD~1 –***This command removes the commit as well as the changes from your working directory. This command can also be called destructive command as we would not be able to get back the changes so be careful while using this command.

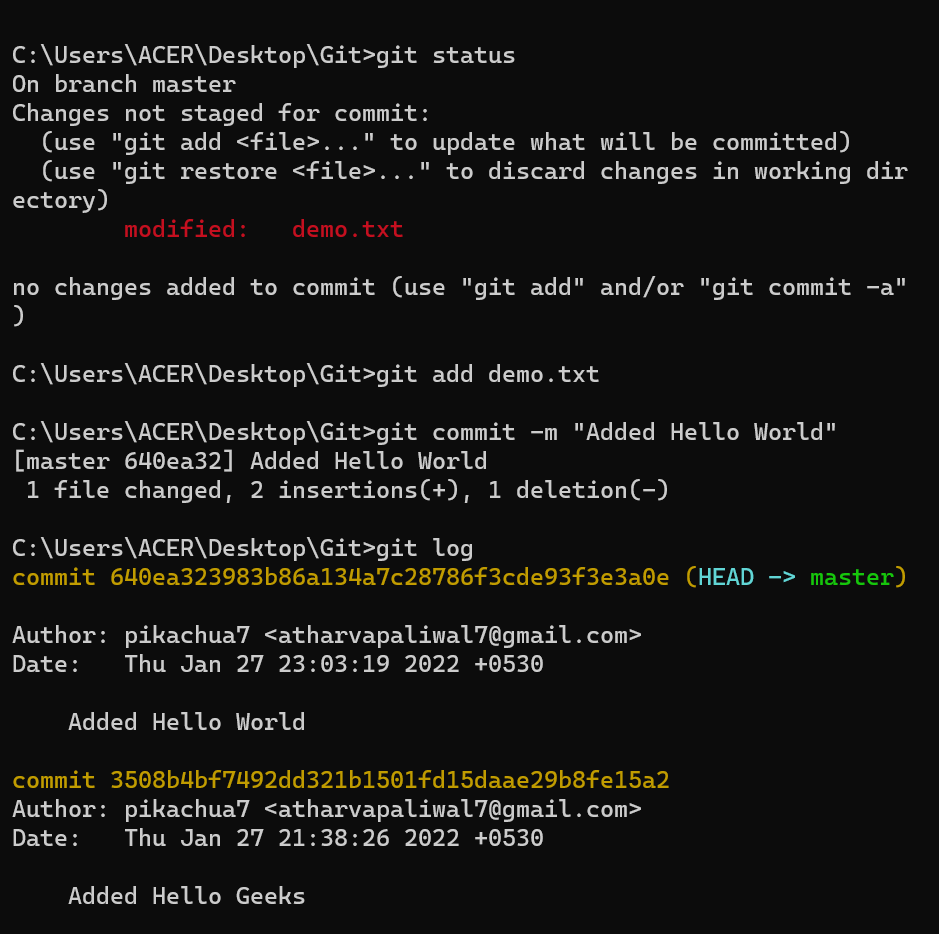
*Points to keep in mind while using git reset command –*

* *If our commits are not published to remote repository , then we can use git reset.*
* *Use git reset only for removing commits that are present in our local directory and not in remote directory.*
* *We cannot remove a specific commit with the help of git reset , for ex : we cannot say that we want to remove the second commit or the third commit , we can only remove latest commit or latest 2 commits … latest N commits.(HEAD~n) [n here means n recent commits that needs to be deleted].*

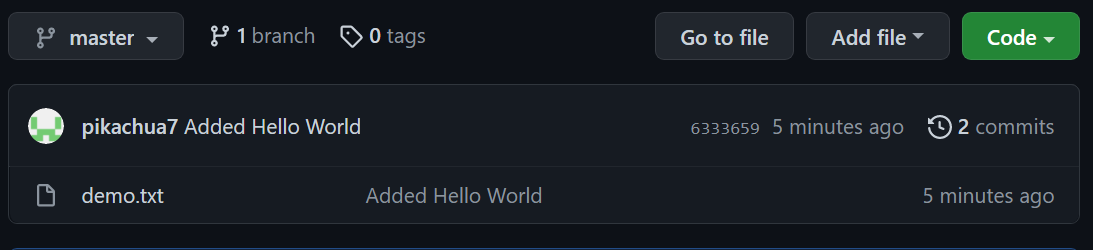
We just discussed above that the git reset command cannot be used to delete commits from the remote repository, then how do we remove the unwanted commits from the remote repository The command that we use here is –

**3. git revert**

git revert is used to remove the commits from the remote repository. Since now our changes are in the working directory, let’s add those changes to the staging area and commit them.



Now let’s push our changes to the remote repository. You can see [***here***](https://www.geeksforgeeks.org/git-lets-get-into-it/) how to push your changes from local repository to remote repository.



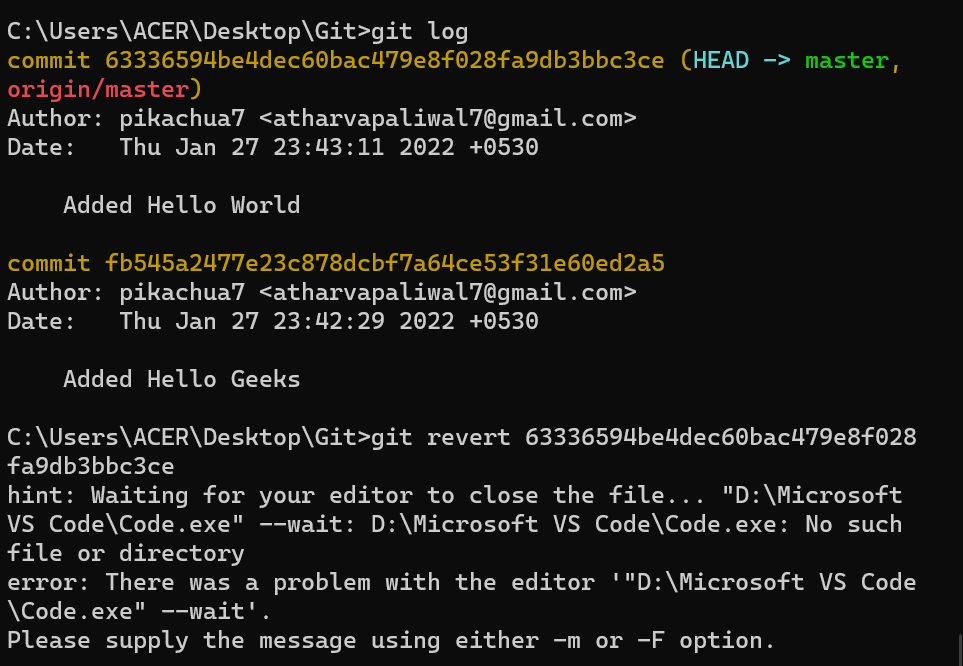
Now we want to delete the commit that we just added to the remote repository. We could have used the git reset command but that would have deleted the commit just from the local repository and not the remote repository. If we do this then we would get conflict that the remote commit is not present locally. So, we do not use git reset here. The best we can use here is git revert.

*git revert <commit id of the commit that needs to be removed>*

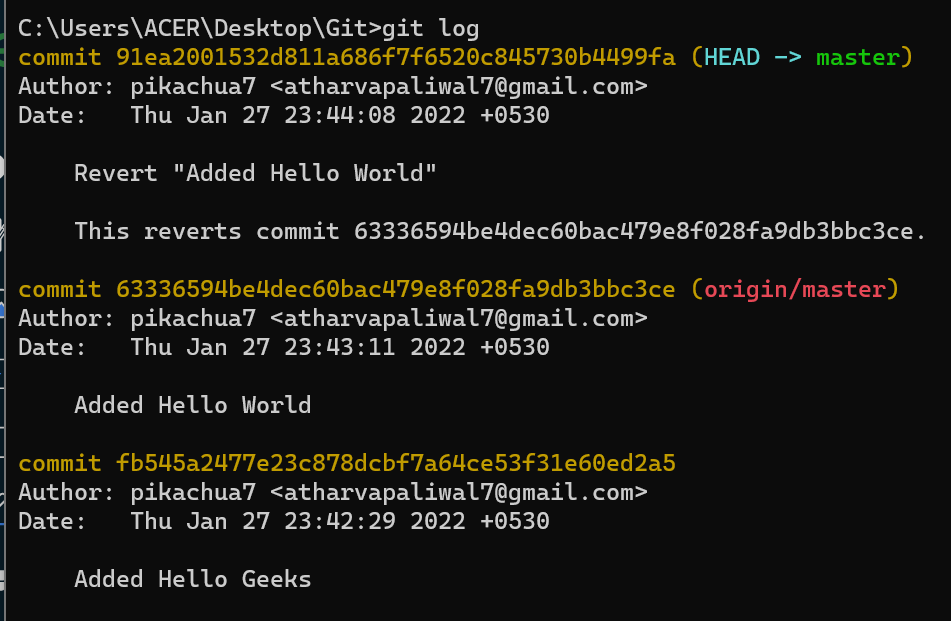
*Points to keep in mind –*

* *Using git revert we can undo any commit , not like git reset where we could just remove “n” recent commits.*

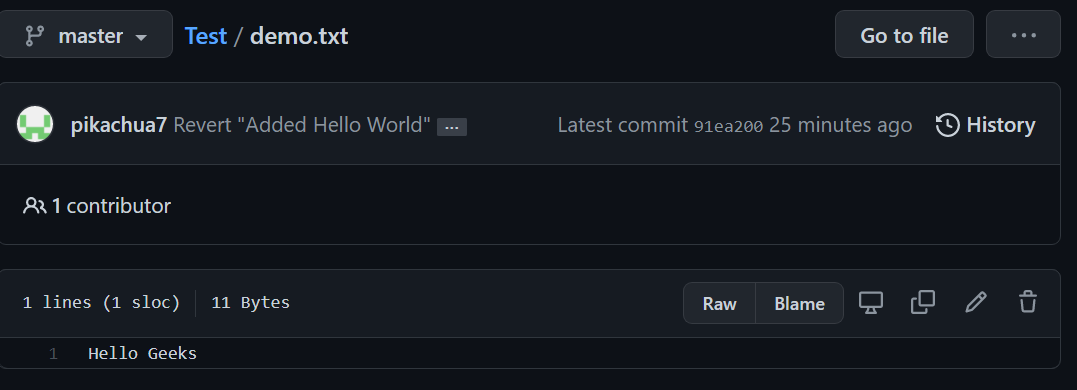
Now let’s first understand what git revert does, git revert removes the commit that we have done but adds one more commit which tells us that the revert has been done. Let’s look at the example –



***Note****: If you see the lines as we got after the git revert command, just visit your default text editor for git and commit the message from there, or it could directly take to you your default editor. We want a message here because when using git revert, it does not delete the commit instead makes a new commit that contains the removed changes from the commit.*



We can see that the new commit is being added. However since this commit is in local repository so we need to do ***git push*** so that our remote repository also notices that the change has been done.



And as we can see we have a new commit in our remote repository and ***Hello World*** which we added in our 2nd commit is being removed from the local as well as the remote repository. Let’s summarize the points that we saw above –

**Difference Table**

| **git checkout** | **git reset** | **git revert** |
| --- | --- | --- |
| Discards the changes in the working repository. | Unstages a file and bring our changes back to the working directory. | Removes the commits from the remote repository. |
| Used in the local repository. | Used in local repository. | Used in the remote repository. |
| Does not make any changes to the commit history. | Alters the existing commit history, | Adds a new commit to the existing commit history. |
| Moves HEAD pointer to a specific commit. | Discards the uncommitted changes. | Rollbacks the changes which we have committed. |
| Can be used to manipulate commits or files. | Can be used to manipulate commits or files. | Does not manipulate your commits or files. |