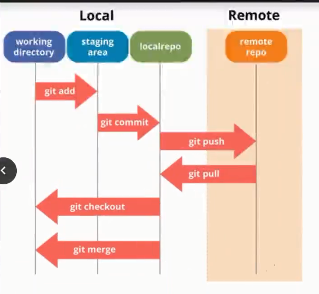
Git Hub

1. Create instance on AWS Ec2
2. Architecture



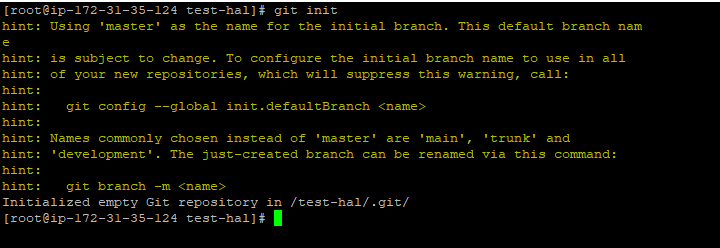
1. Install git on it.

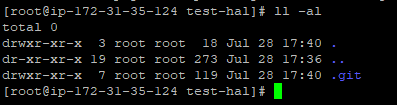


1. Create test folder as shown in given location



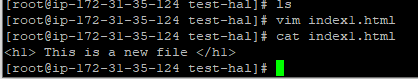
1. Run “git init” to initiate git environment



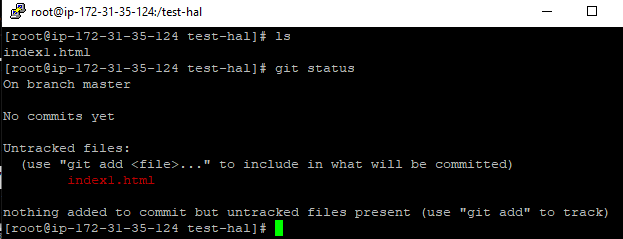


1. Create a Test file “vim index1.html”

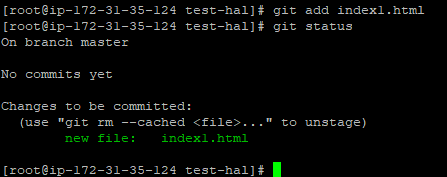




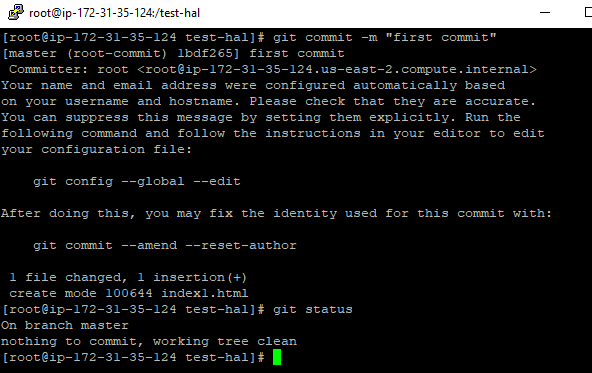
1. If you run “git status”. It will show that file “index1.html” is in Untracked file.



1. To add file to environment Run “git add index1.html”

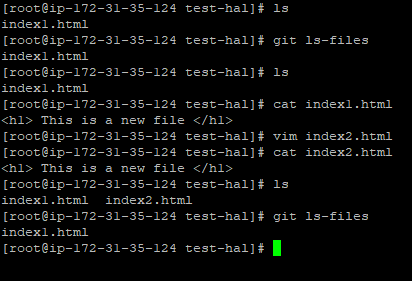


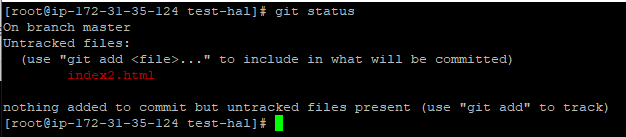
1. Now as we have added this file to staging area. Now it is our local working directory. To get file moved to github we need to commit our changes with command “ git commit -m "first commit" ”



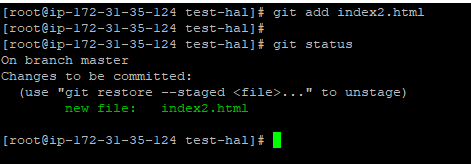
1. After we commit the file, it git-ls-file show file in local repository and ls will show file in working directory. After we git add files like index2.html than it will go to local repository



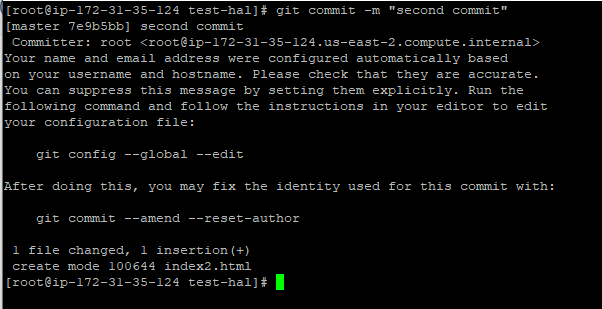


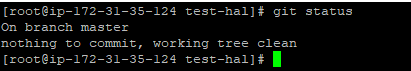


1. Run git add to add file to staging area,



1. Commit the changes

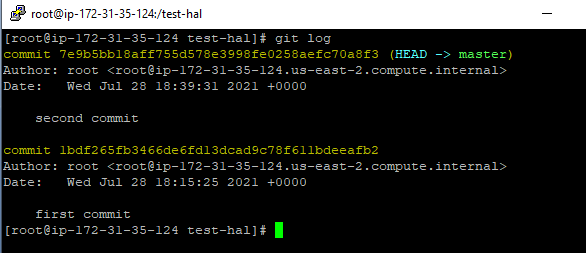


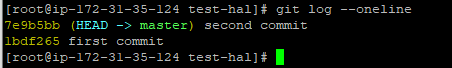


1. After this our file will be in local repository.

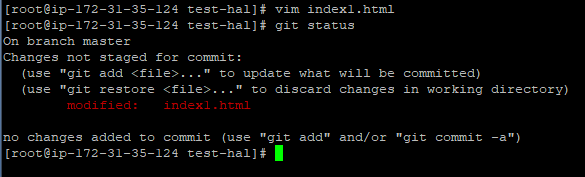


1. Everytime we commit it generate commit id.

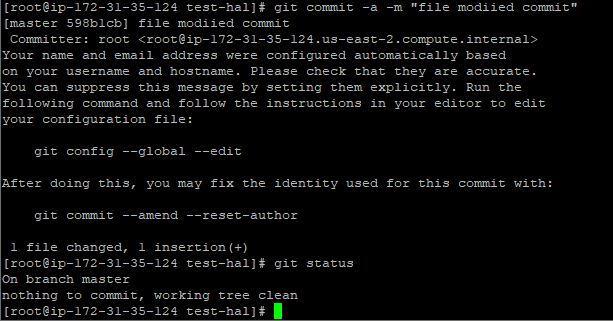




1. Let us check to modify file. Here you will see it is showing modified instead of untracked. Changes we did are in working directory.

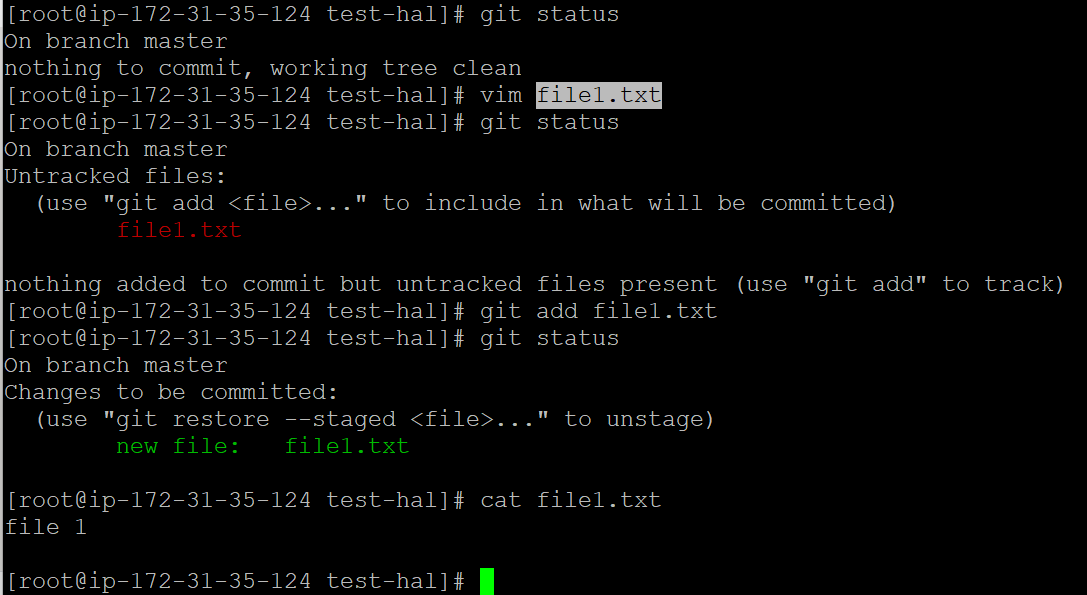


1. We will commit finally



Let us create another file

File1.txt



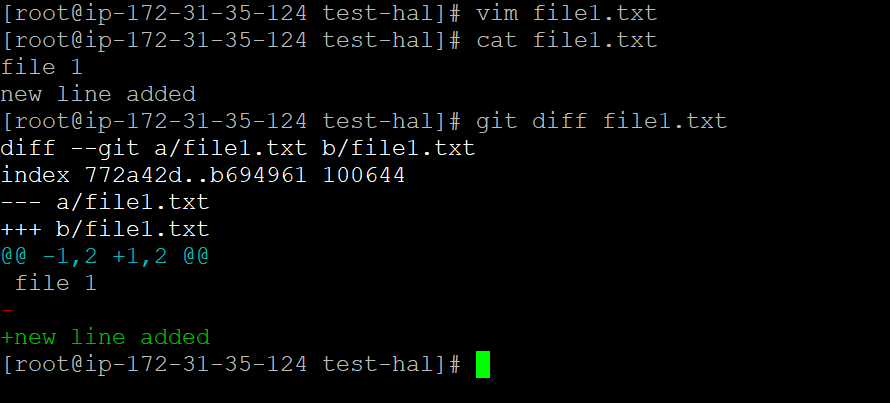
Now I would commit this file



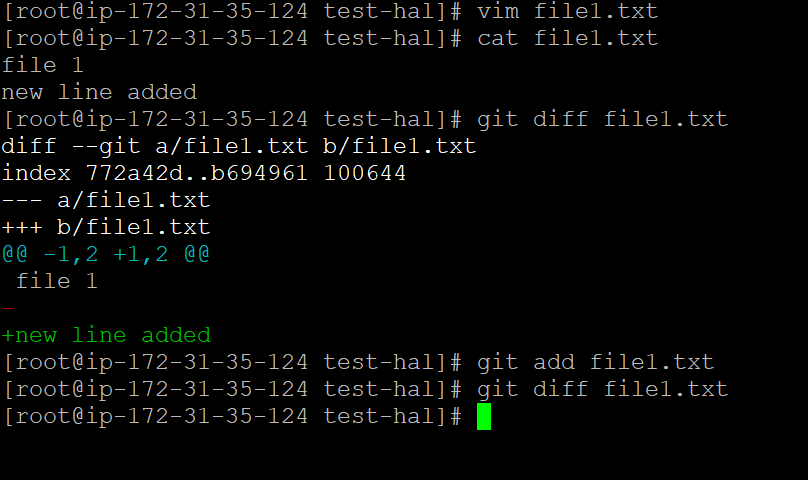
Now let us add some test more in file1.txt

After I do this it is still in working directory. Now this file changes will be called in unstagged changes

To check unstagged changes we use git diff file1.txt command as shown below:-



Now if we add this file to staging area git dioff will not show the above output.



To see the staged changes of a file we will run command git diff –staged file1.txt



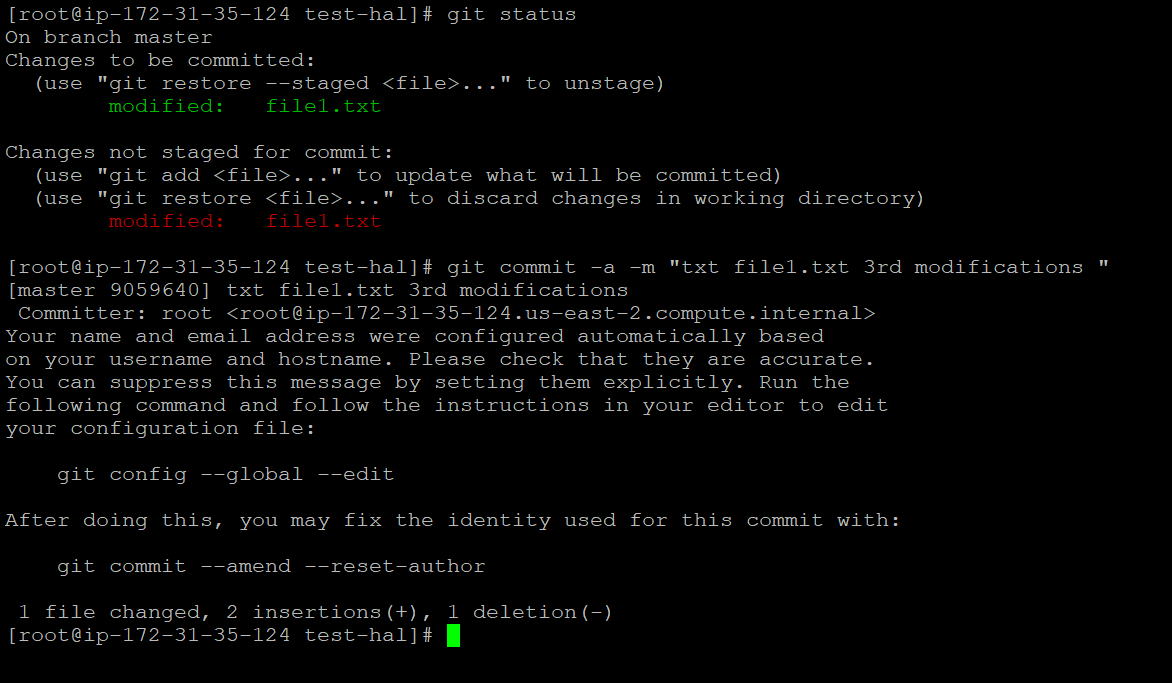
Let us add one more line to file1.txt

This file will show you unstagged change if we run diff command so below is snapshot to show changes output as per commands



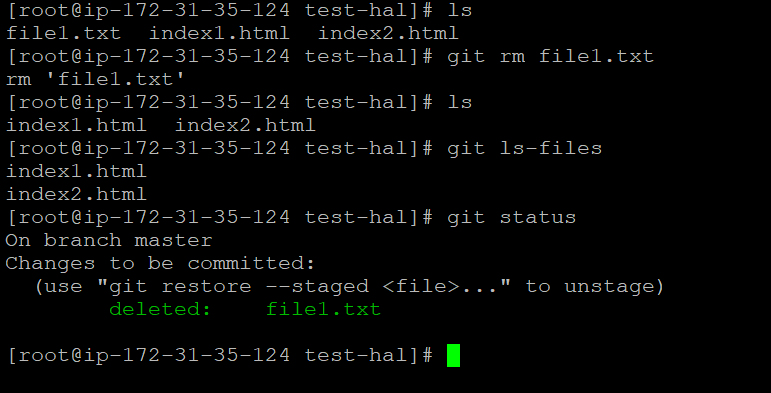
Git status will give us details of staged changes and unstaged changes.

But diff command help to know staged and unstaged changes specially

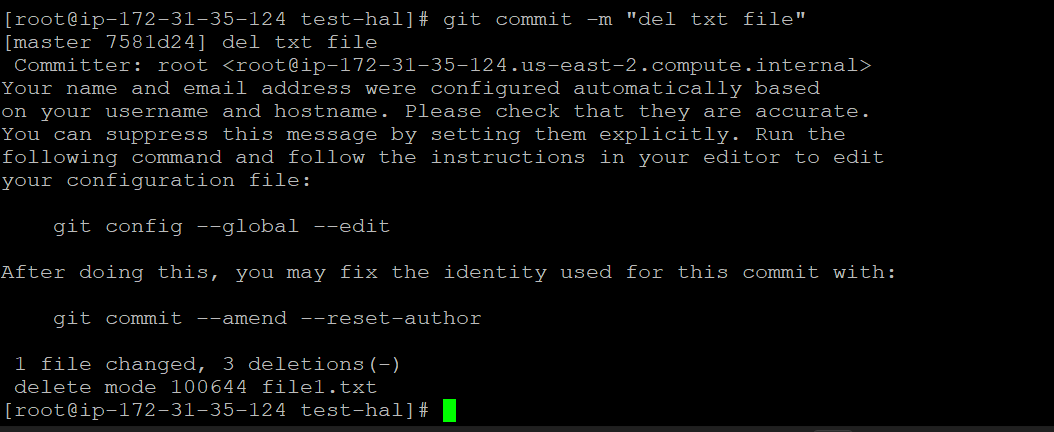


Here –a also add and commit in one command

To delete the file we will run

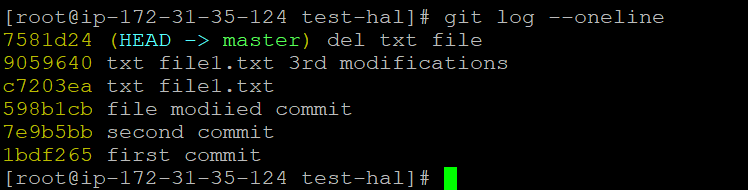


File will be deleted from both working directory and local repository

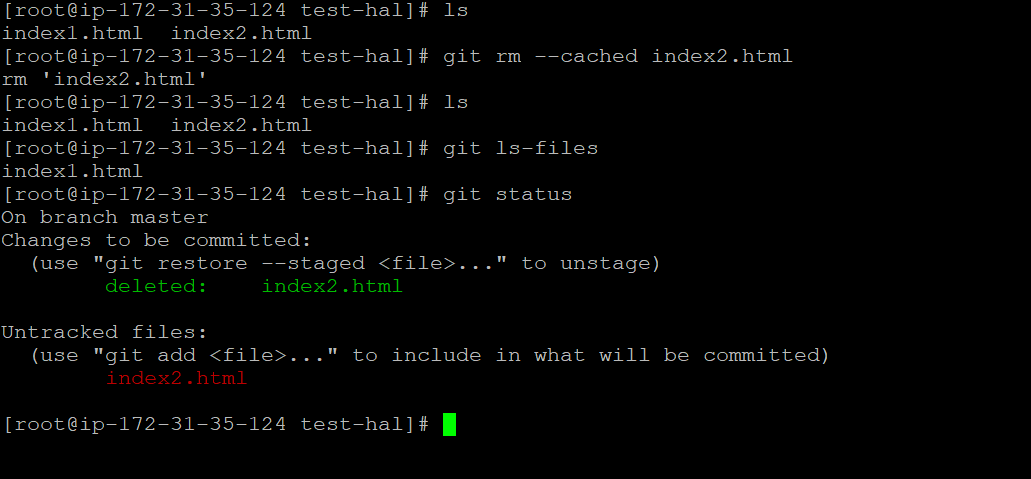


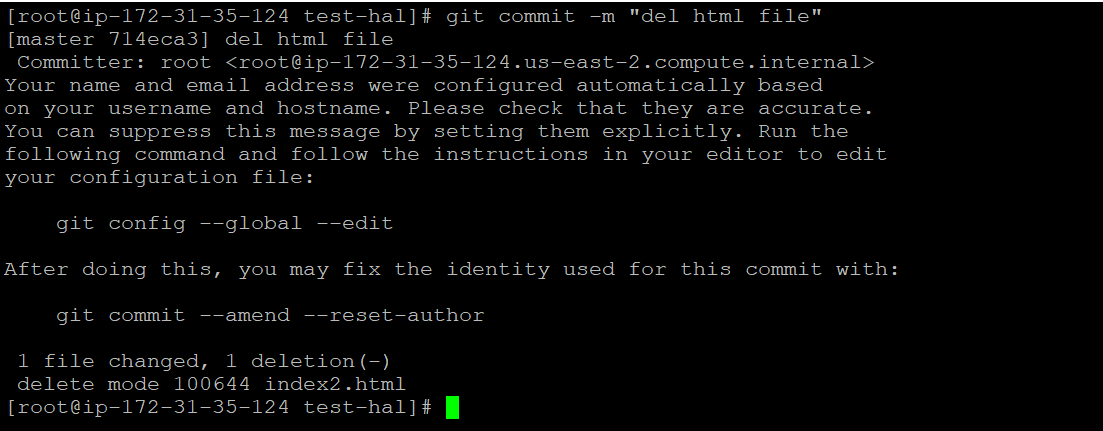
To check commits as per commit id we will run

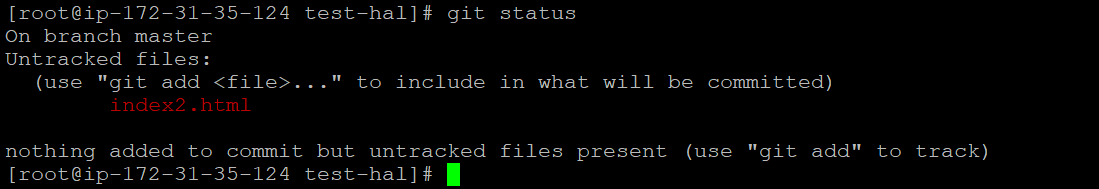
Git log –oneline



If we want to keep file in working directory but we want to delete it in local repository than we will do below commands

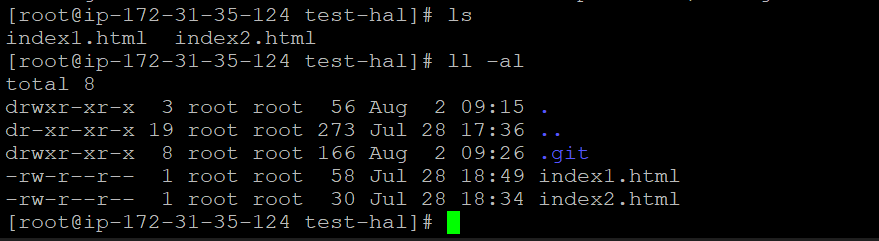


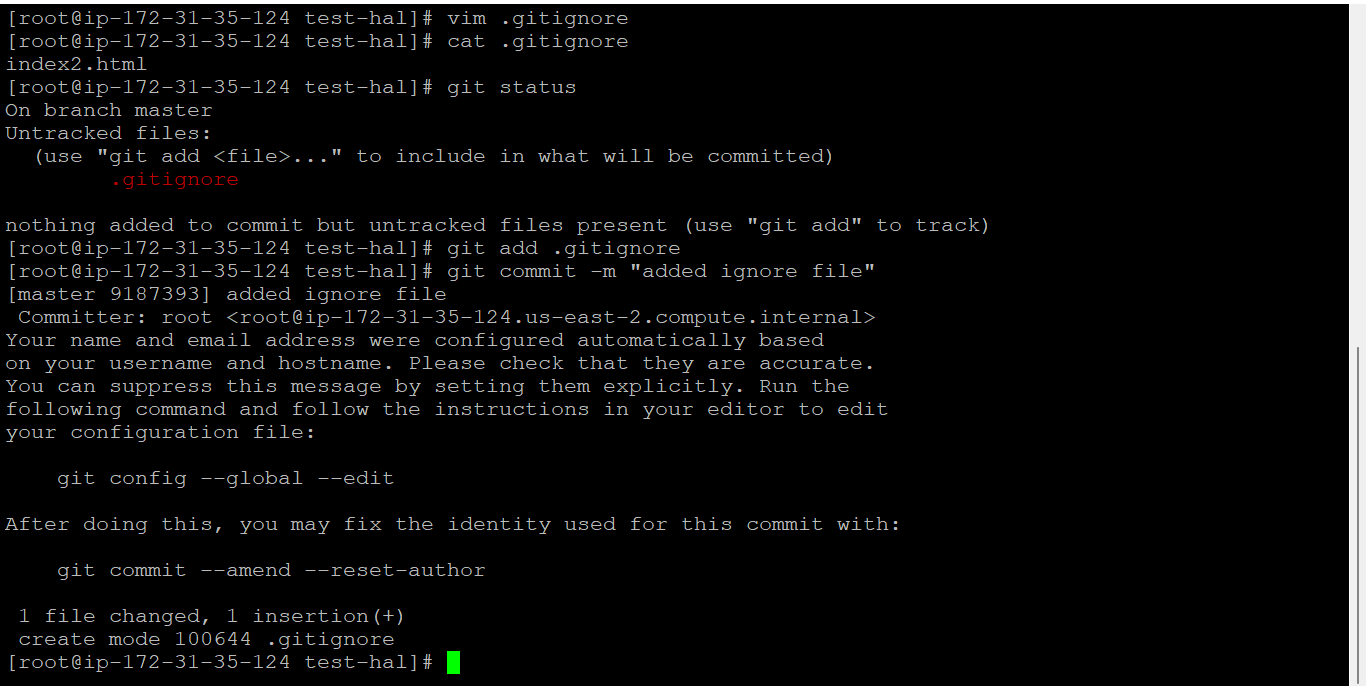


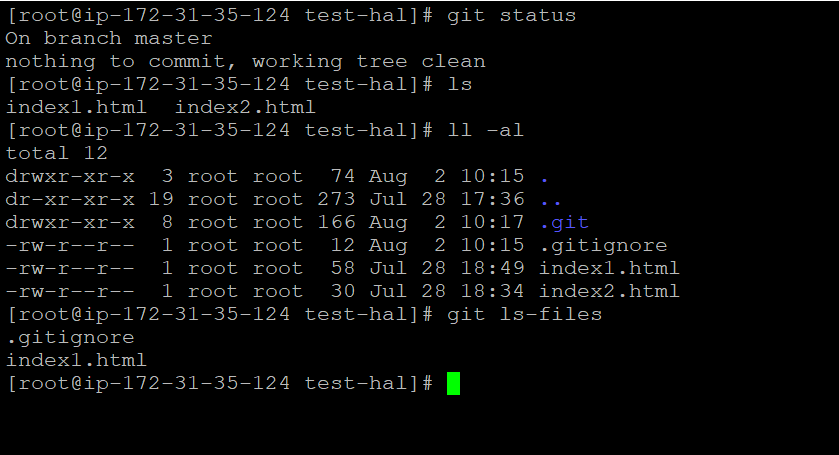


Now we see index2.html in untracked because we have deleted it from local repository and kept it in working directory.

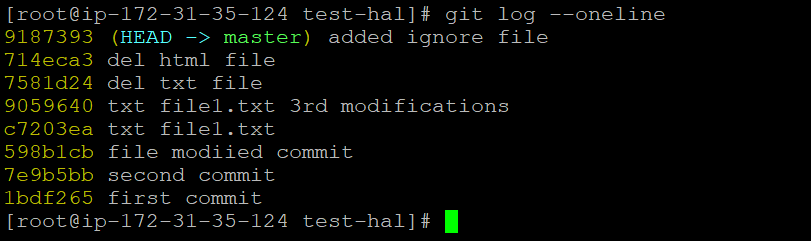
To avoid this condition when we want to keep file in working directory but not in local repository than we will use .gitignore



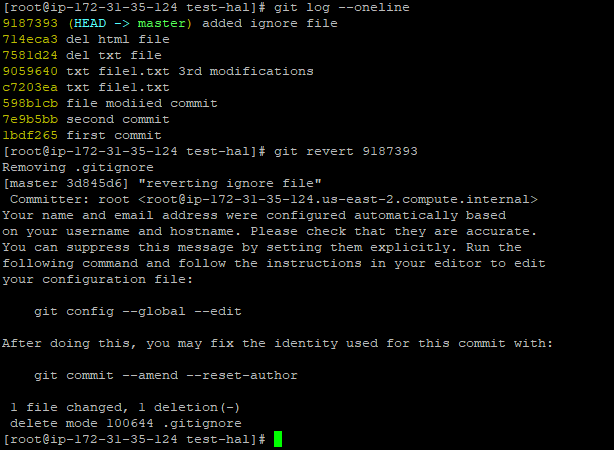




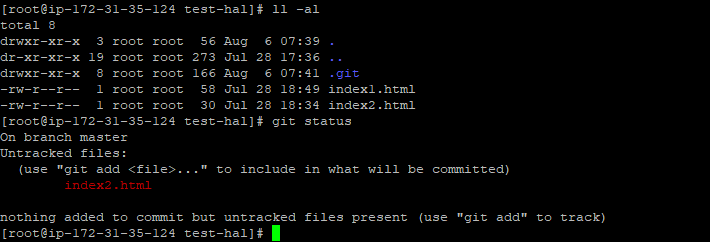
To check commit ids

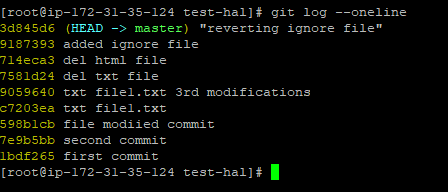


If we want to undo any commit change than



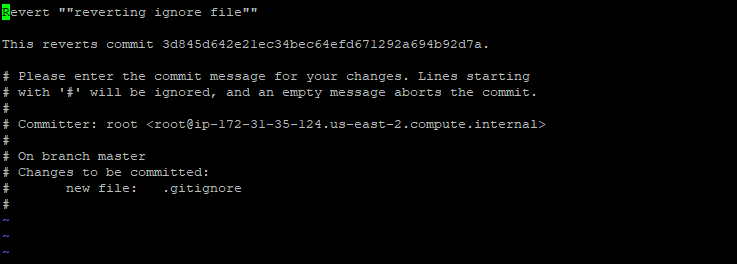
Now as we have removed gitignore file index2.html will again become untracker file as shown below

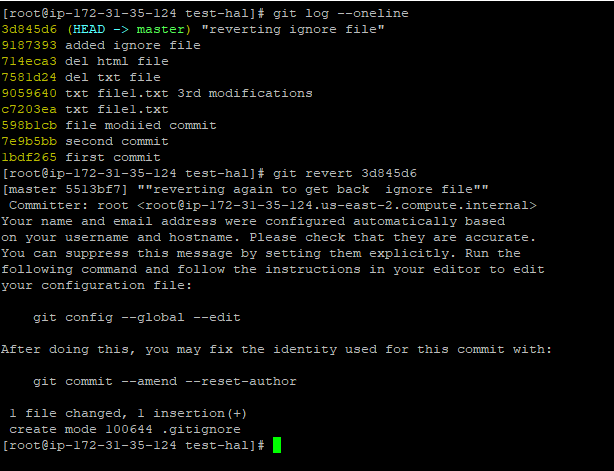


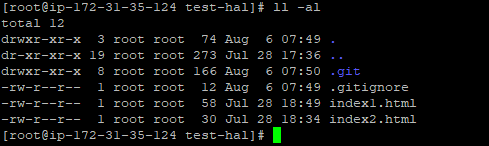


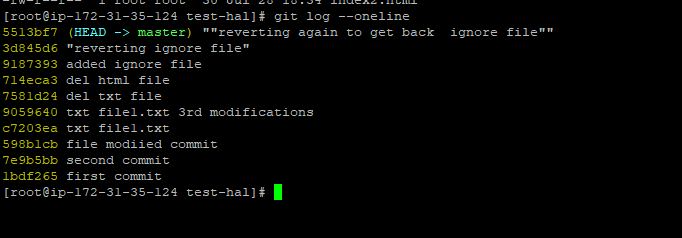
As you see above log has generated ne commit id for added ignore file and deleted ignore file

Let us revert again









Another way to undo our changes

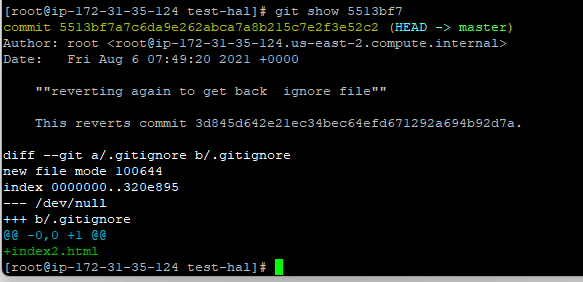
If we use reset instead of revert than all commit will be gone and log stamps will be gone.

Commit after id of reset commit will be deleted from history also.

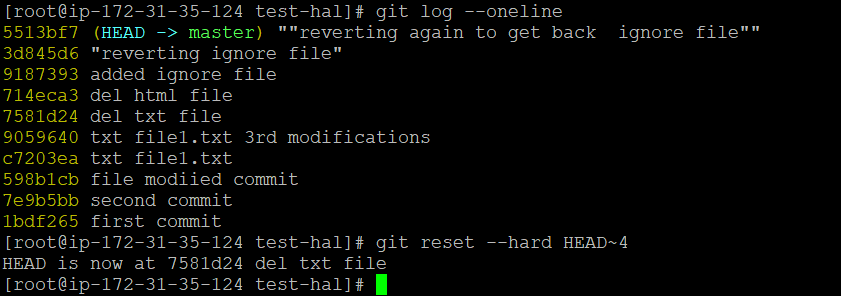
Reset will not generate any commit id, it will will also not let anything exist in log history.

|  |
| --- |
| Example  #git reset –hard HEAD~2 or git reset –hard ibdf265 |

To see all commit activity in git id we use git show command

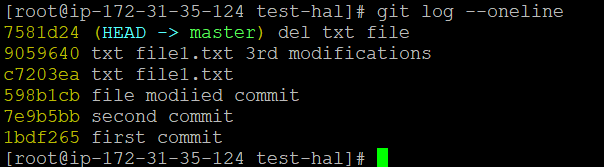


Now using reset we can set HEAD to position which we want to choose for example as shown below



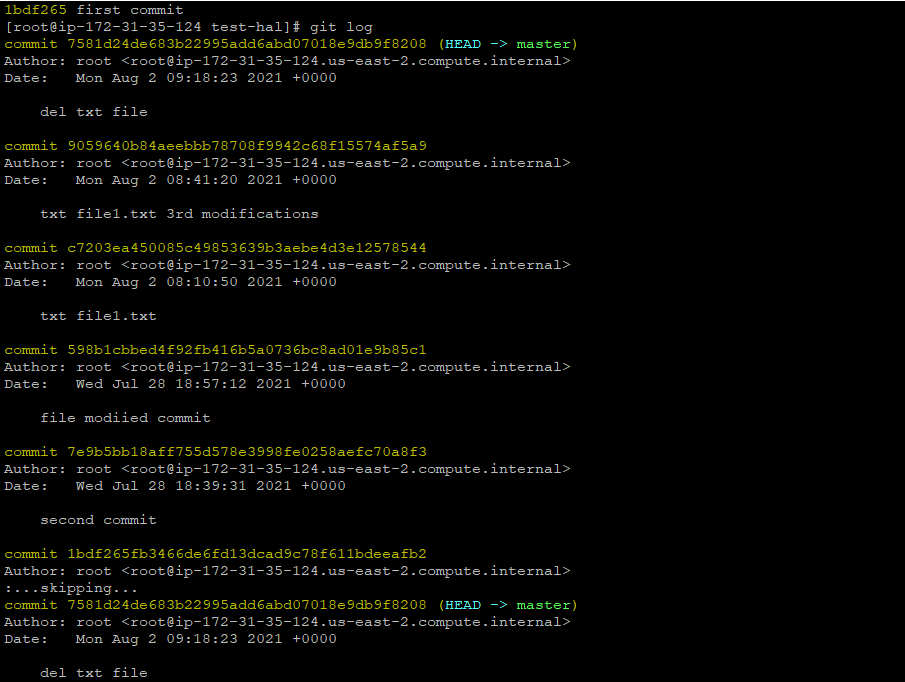
Here the id is set to 758d24

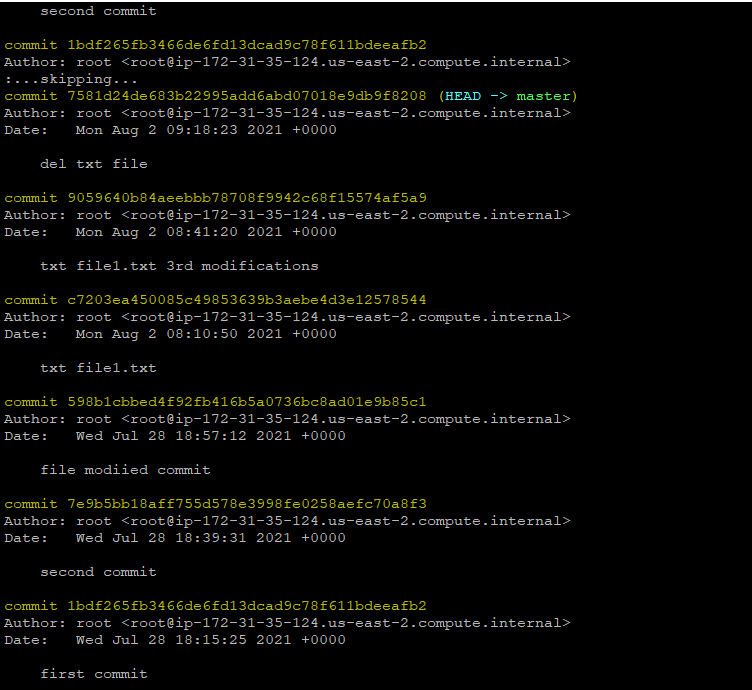
This way we can change the header .



In above you see old history is gone, now head is reduced . Now you cannot get back those commit, it is completely removed from database.

With git log command you can see all commit id with time stamp





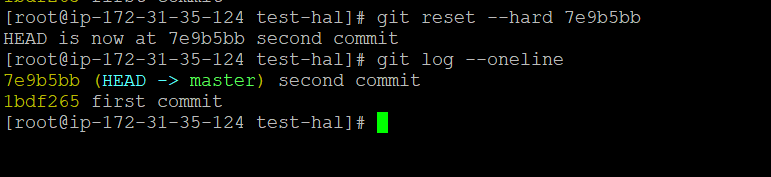
Note- reset is a destructive command.

Concept of Branching

Our default branch was Master branch

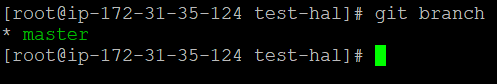
Let us keep only two commit now to understand branching in easy way

I am removing other commit too. We can use both HARD~4 or id of commit as shown above and below

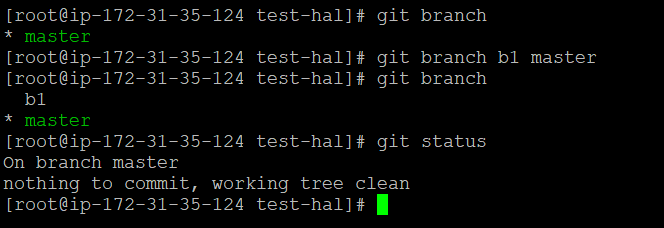


To check your current location in branch we use

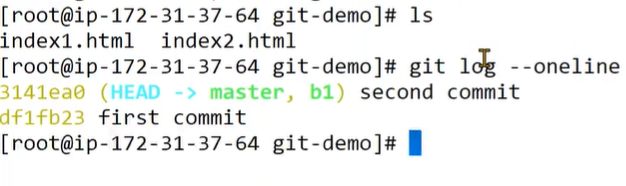
Star will show your branch location



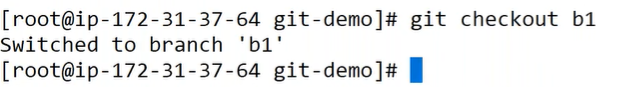
To create sub branch for master we use command as shown below

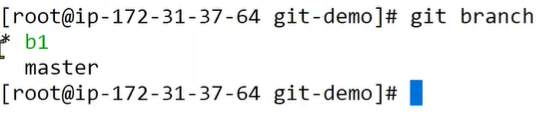


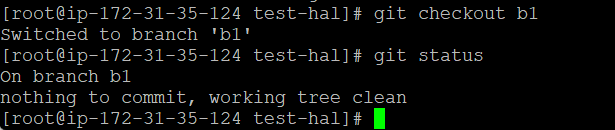
Currently we have only two files

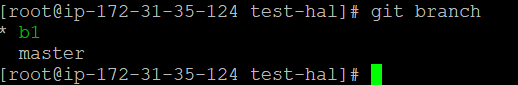


Switch the branch to b1



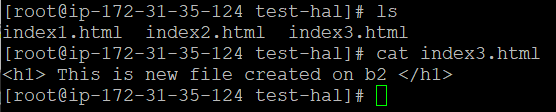


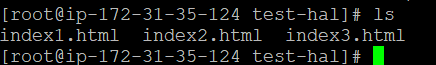




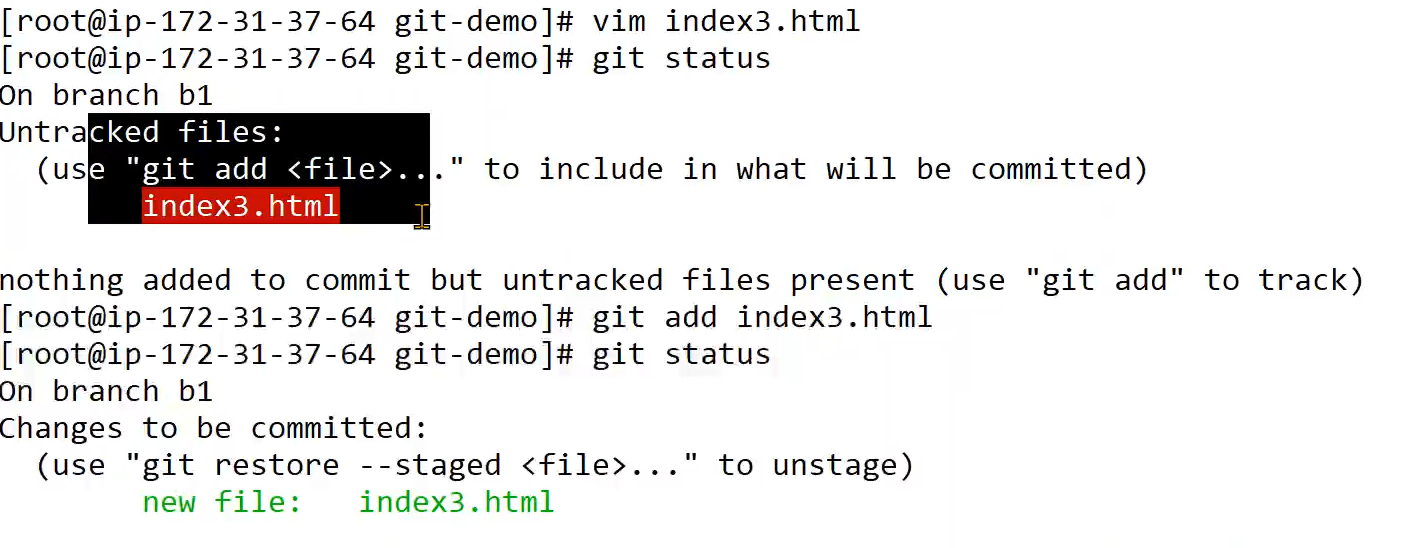
We have reached to branch b1 \* & status command confirms that

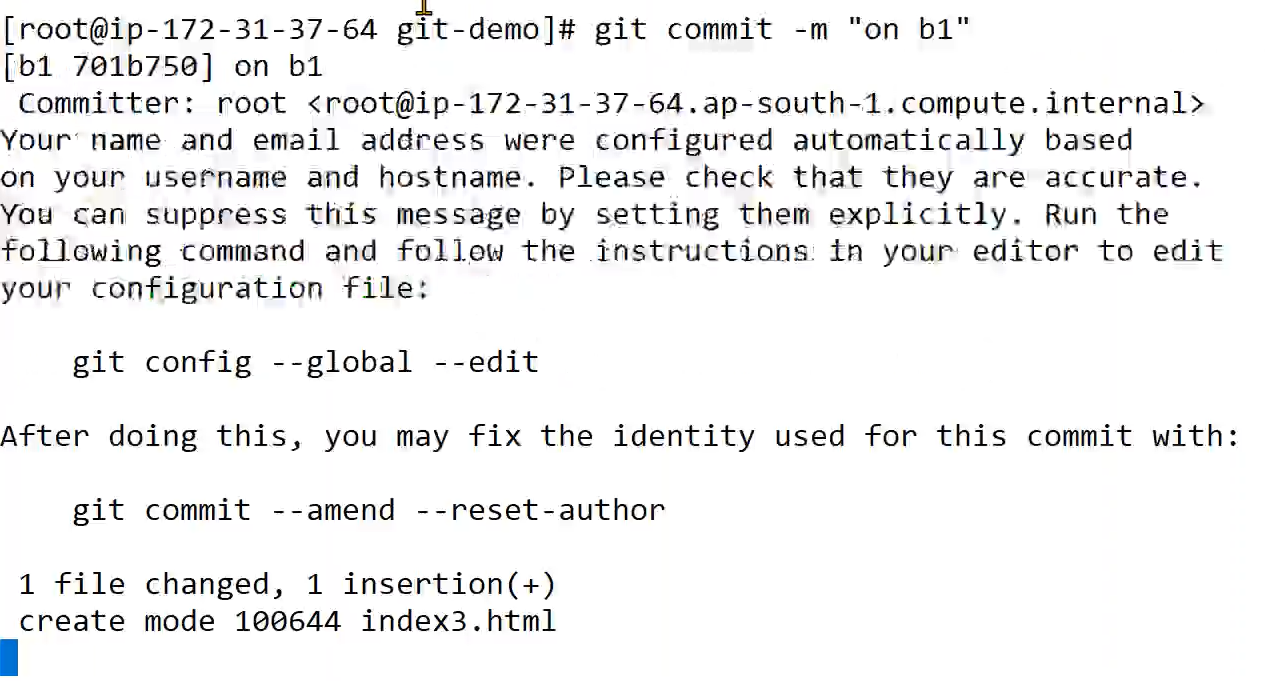
Let us create a file index3.html on b1 brfanch

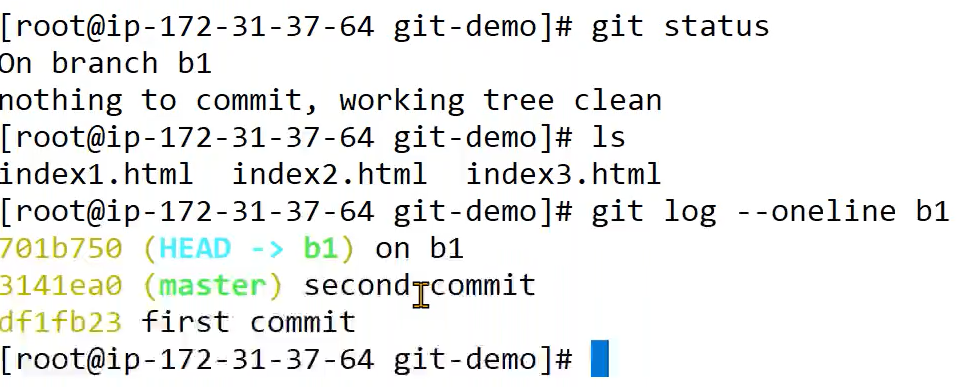




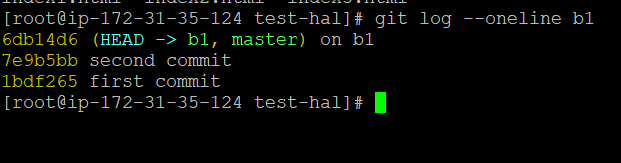
Let us add & commit

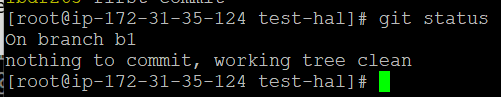






Hence after commit we got branch clean.

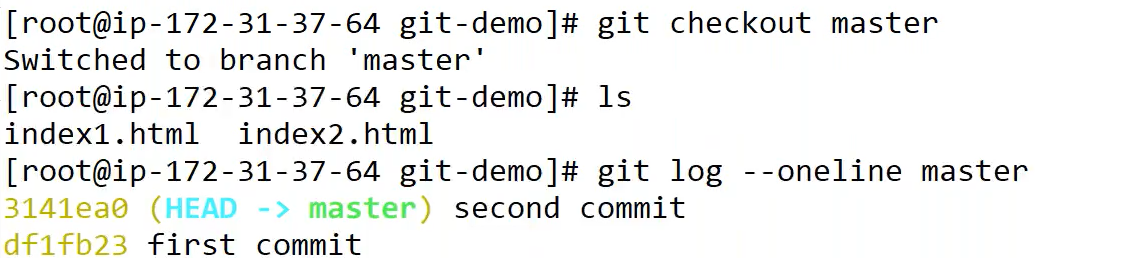




The third commit is not yet available on master branch, it is available on sub b1 branch.

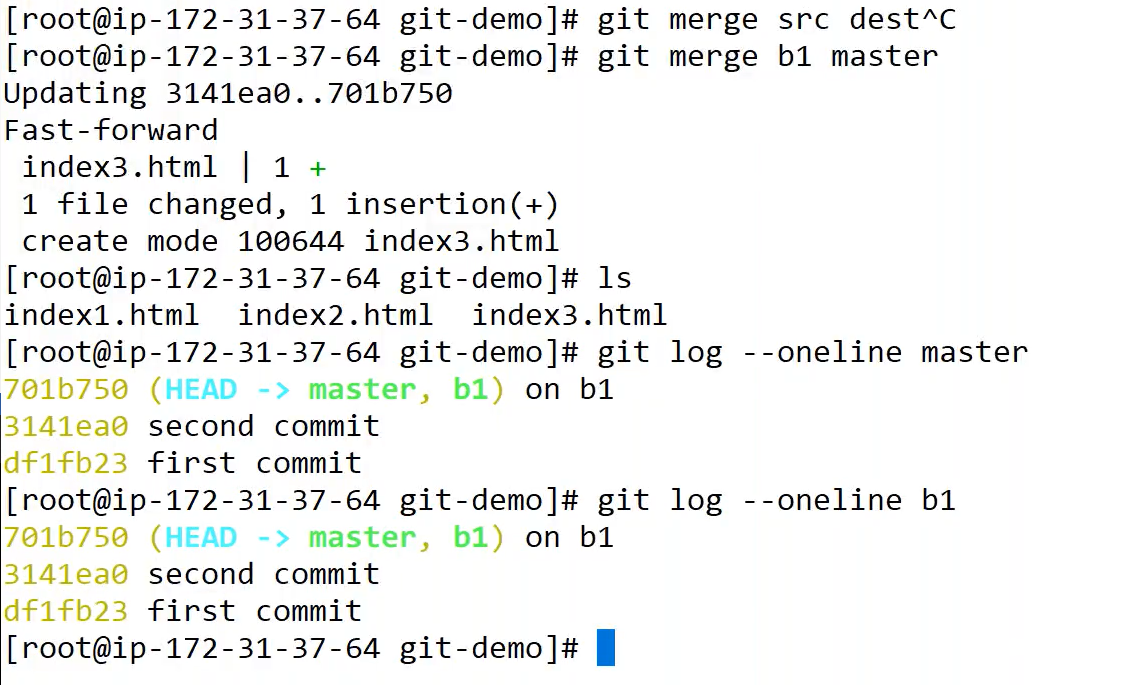
Now to get it available on master branch we need to do below steps



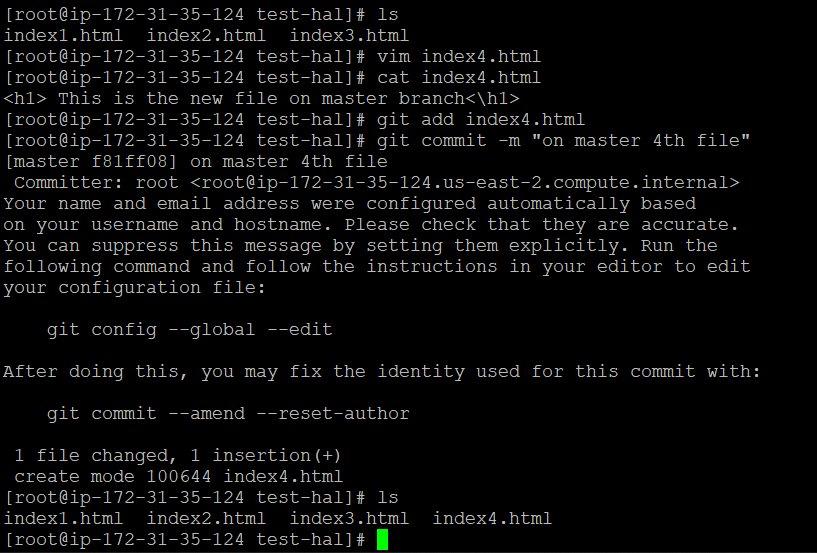


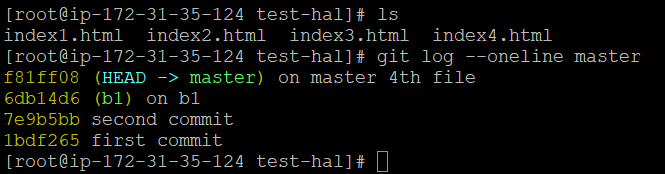
Now the concept of merge branch will be used to make index3.html file available on master.

Git merge source-branch destination-branch

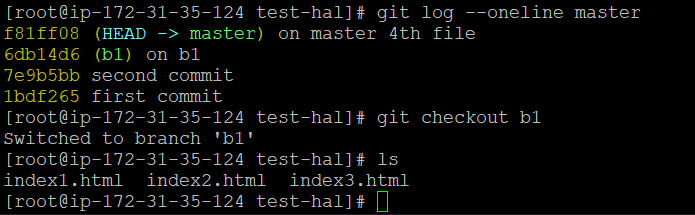


Let us create one more file on master index4.html

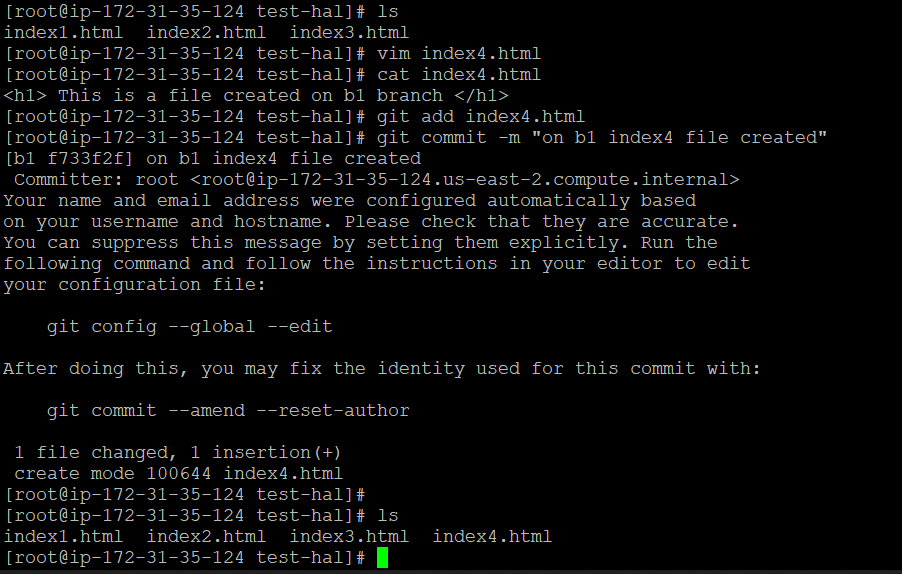




We don’t have this file on b1

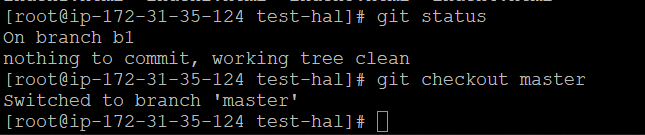


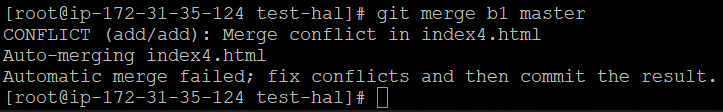
Let us create same file name on b1 branch



Let go to master branch ,

To integrate changes from branch b1 to master

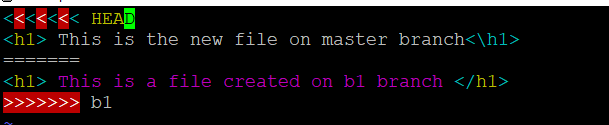




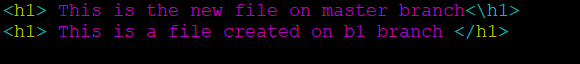
As you see two same file names of different branches shows conflict while merging , now we will remove the conflict. 1:46 min

To resolve conflict

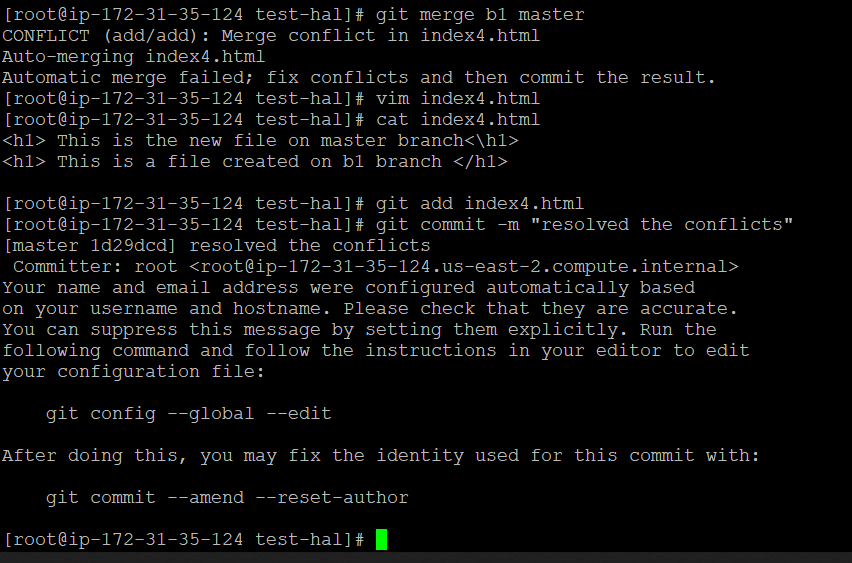
Open the file



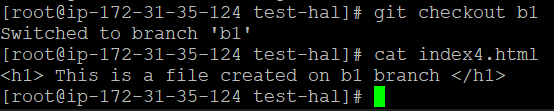
In above conflict file remove all unnecessary data as show below



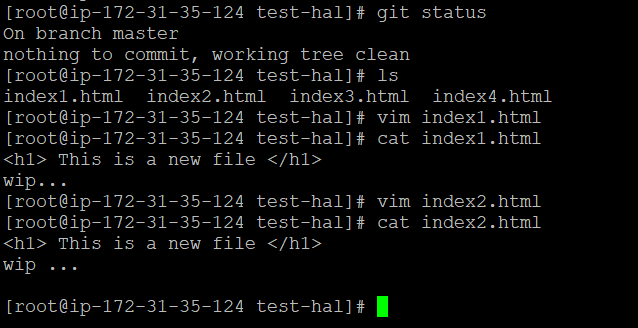
Now commit and try merge again

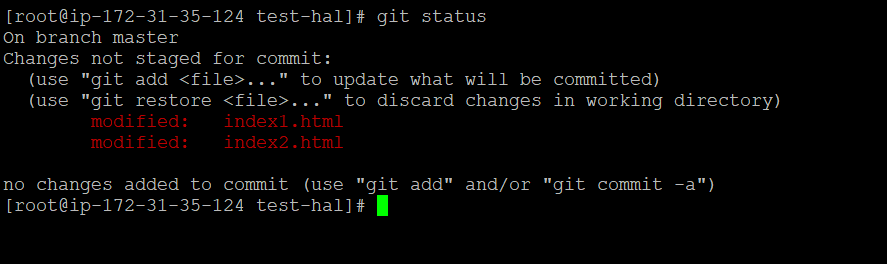


There will be no change on file of branch b1

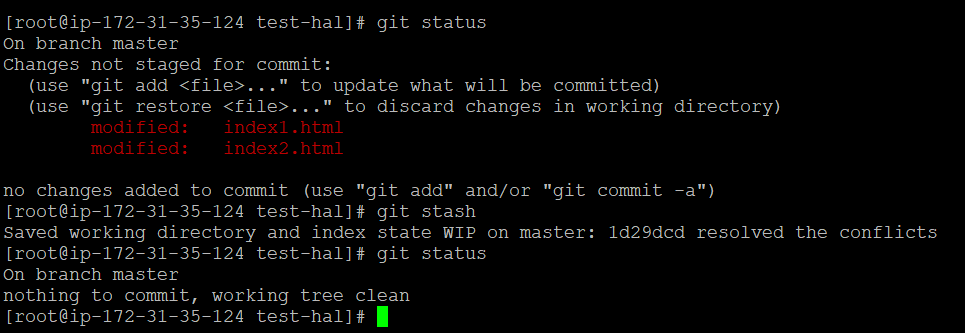


If you are working on any file and work is in progress for example index1.html & index2.html

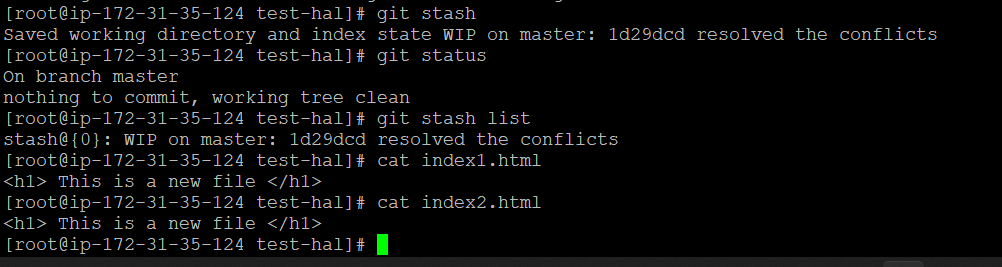




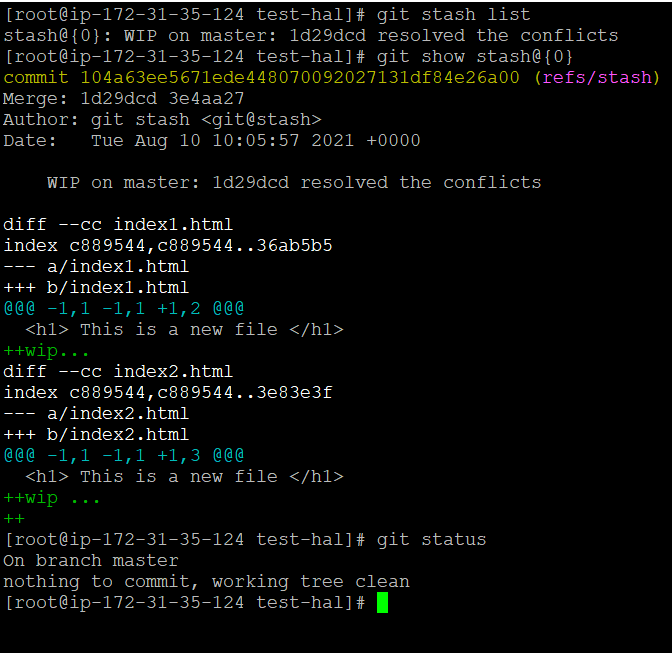
If we want to commit and still our 2 files are incomplete than we have option to bypass these file . to bypass our currently used files we will use stash.



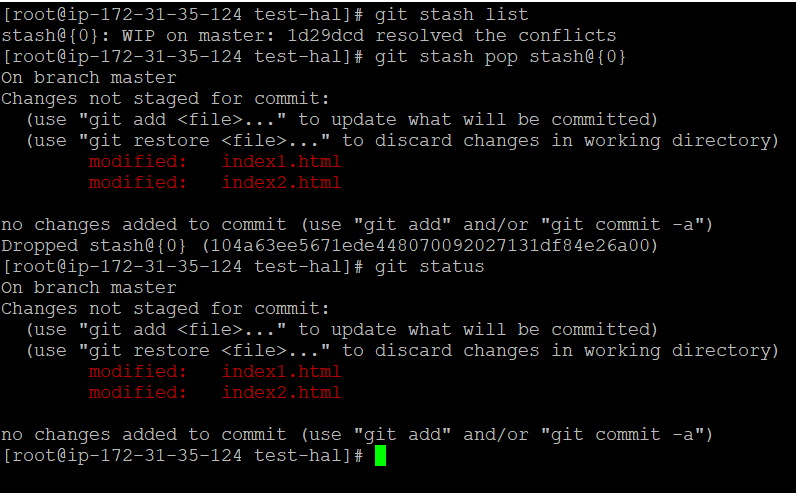
If we see index1.html & index2.html than we will now see wip line as previously , that is moved to stash temporarily .



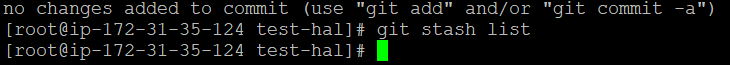
With the help of stash id we can see the changes which are stashed and in work in progress.



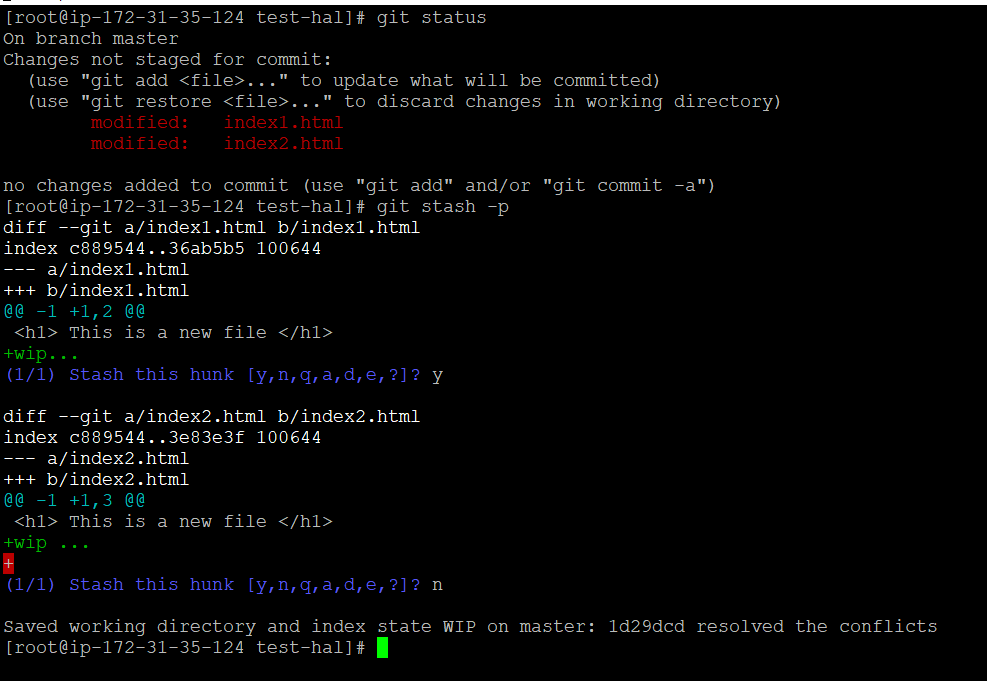
Now by running below command using stash id we can unstash the changes .



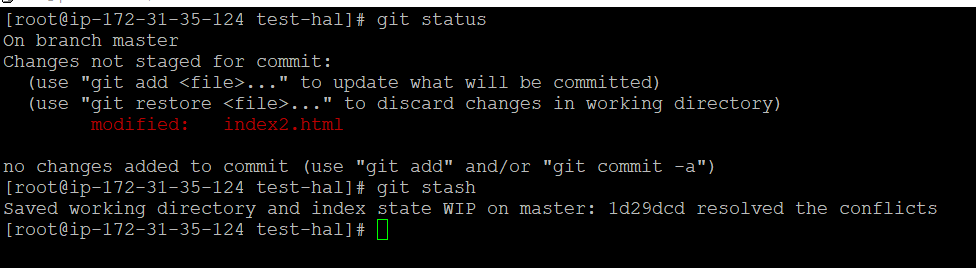
Since we have unstashed the files we will see nothing in list now

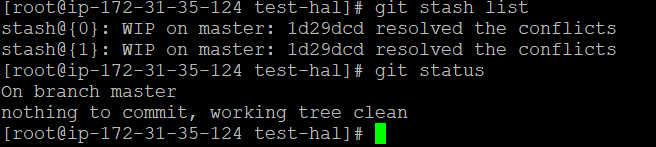


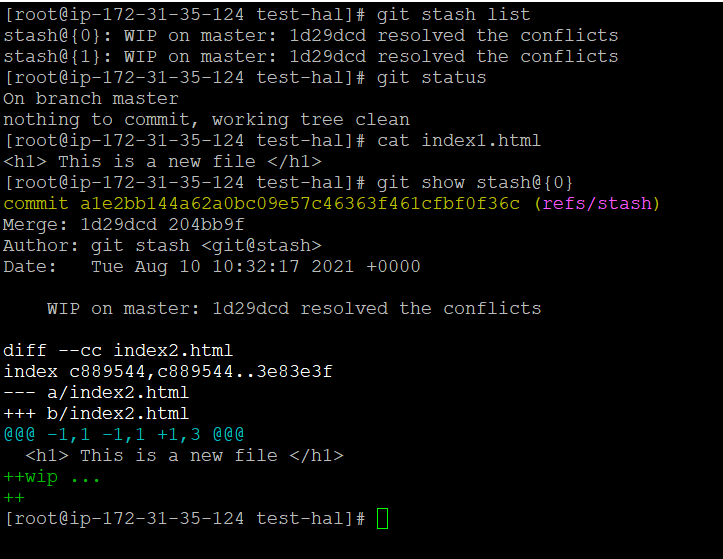
If we want to partial stash means any one file than we will use git stash –p

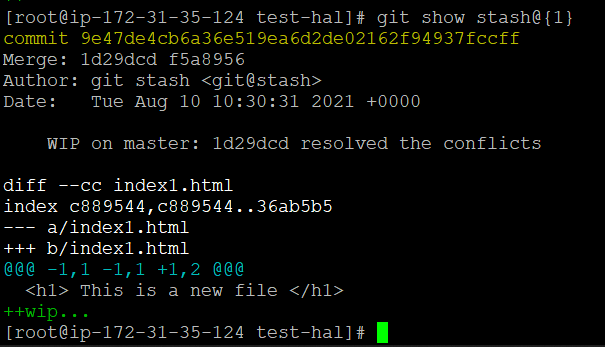


Now we have stashed index1.html but not index2.html



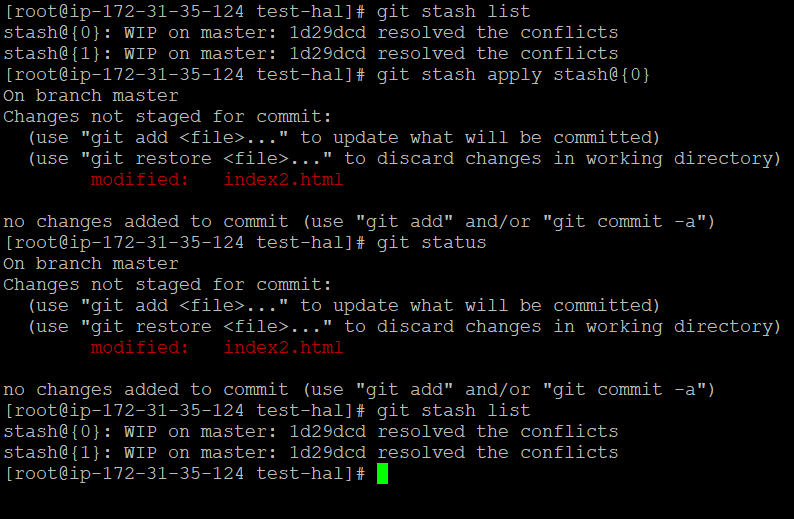


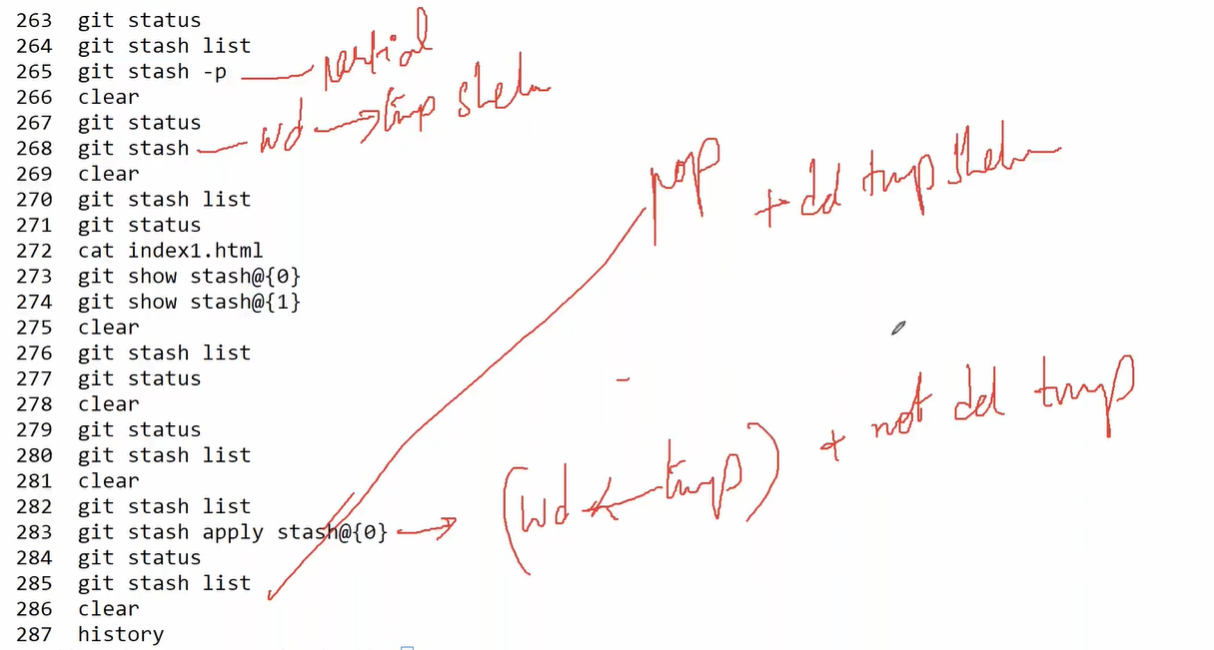




Whenever we stash the git will generate git stash id with the help of which we will get all details of the system at that stash point.

Now again to unstash we will use below command pop( pop delete filw from temporary shell and apply keep file in temporary shell and show us data in unstashed form too).

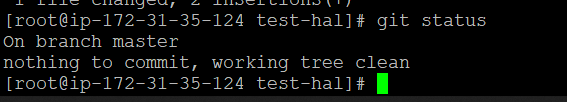




You see above stash list still have the stash id even after apply.

Let say we don’t want to unstash without deleting from temp shell than,



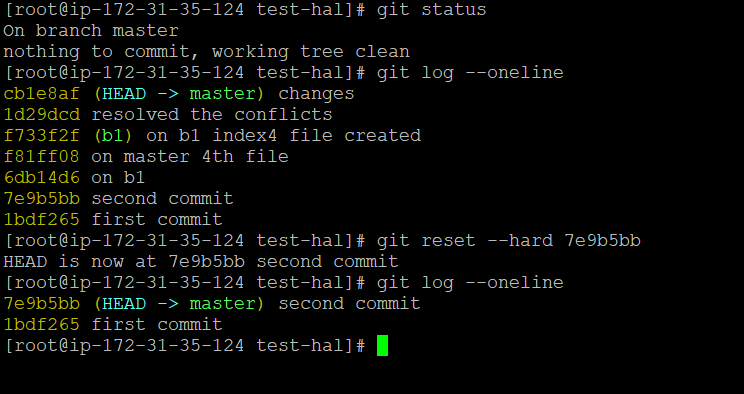


Stash clear will clear all stash in one shot, if we use drop with stash id it will drop only that particular stash .

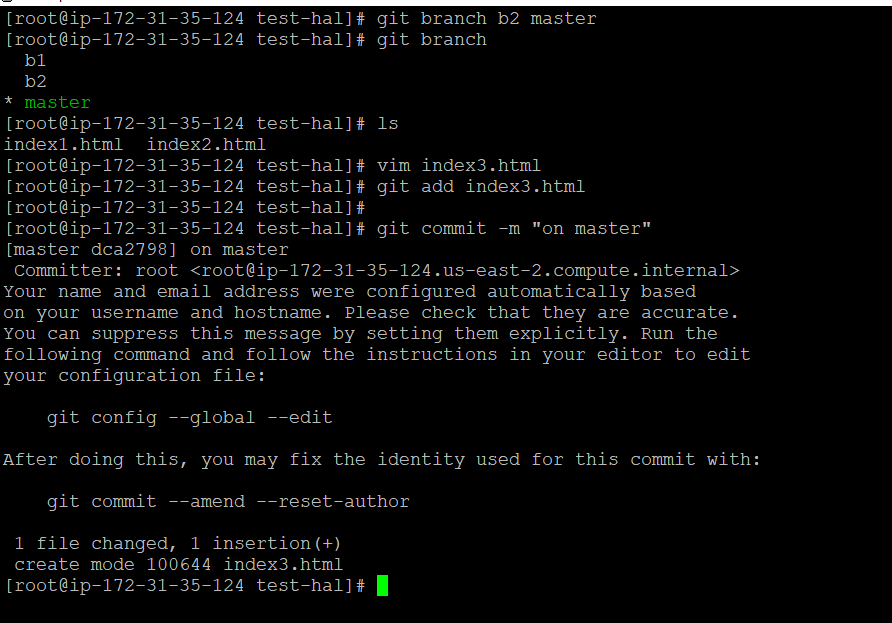
We have deleted file without unstash in above case.

Now let us discuss git rebase (it is kind of merge but it will not combine all commit in one log but it will help to get changes log will be linear and parallel.

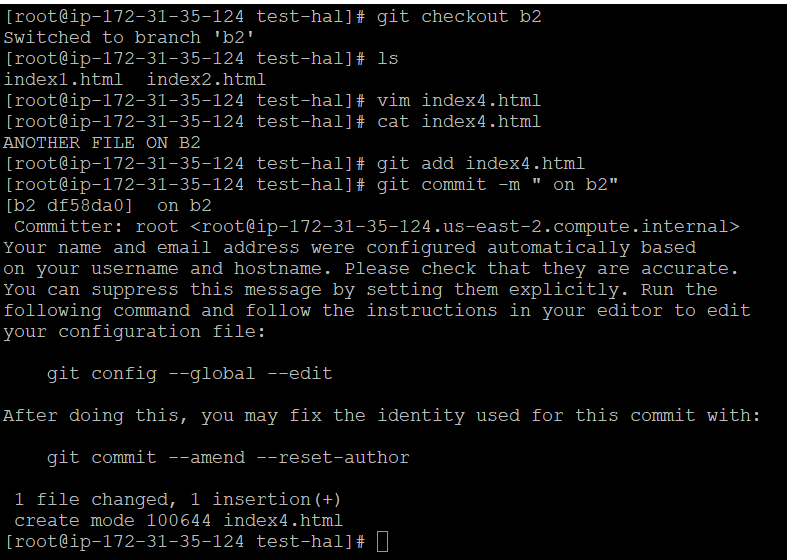
Let us clear old commit and start using only two commit in history as shown below,



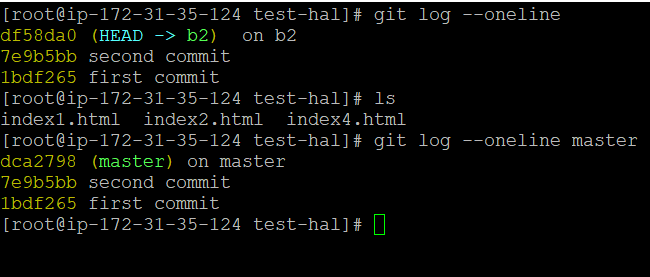
Let us create 2nd branch, and one file index3.html on master branch



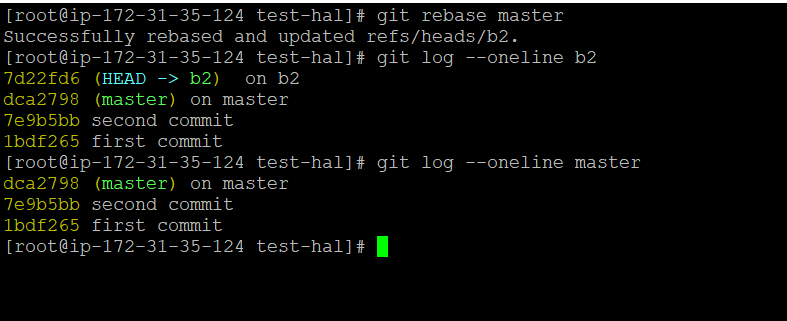
Let us go to branch b2



Now below see log of b2 and master

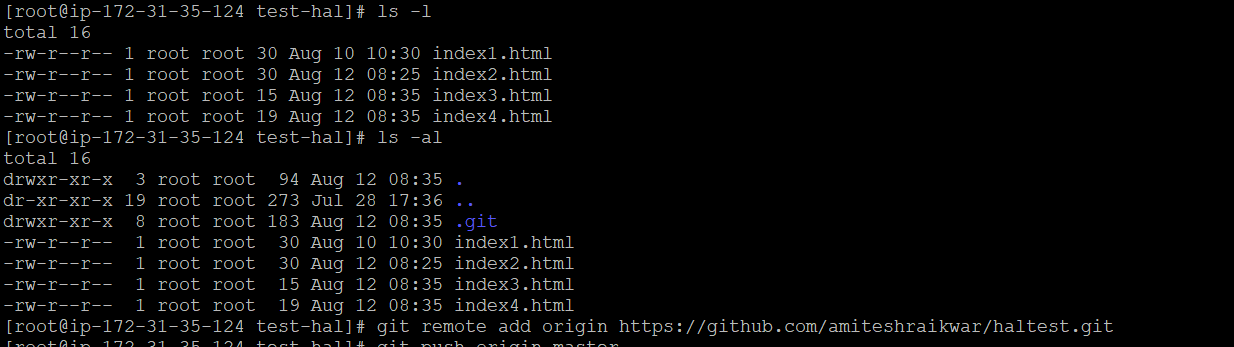


Generally we rebase on child branch

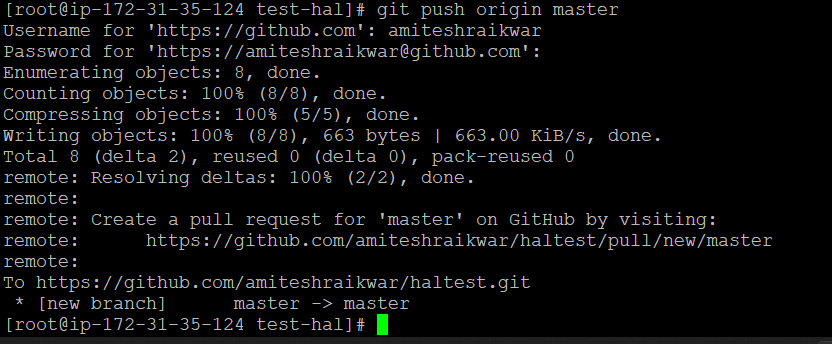


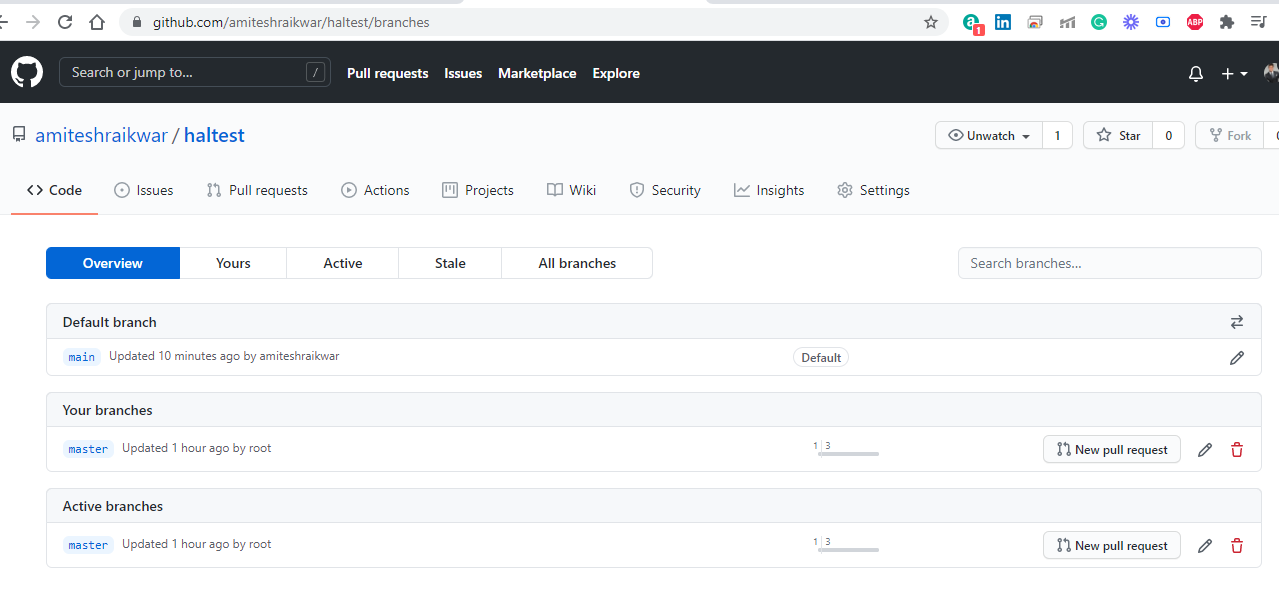
Now let us push these files to remote repository

Before that let us add our url to workstation as shown below

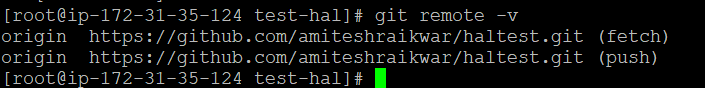


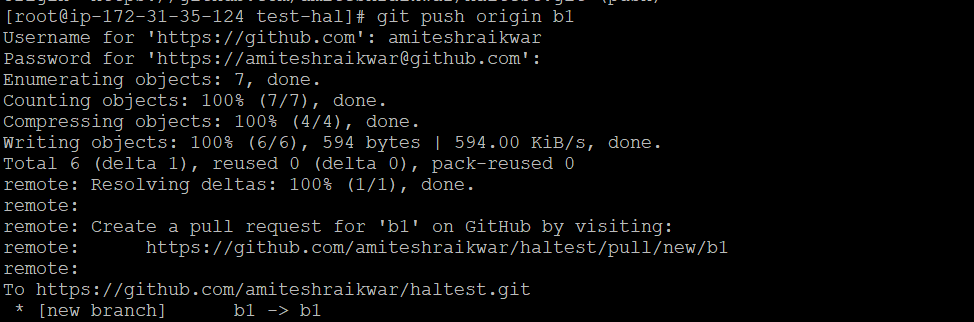
Now we will push





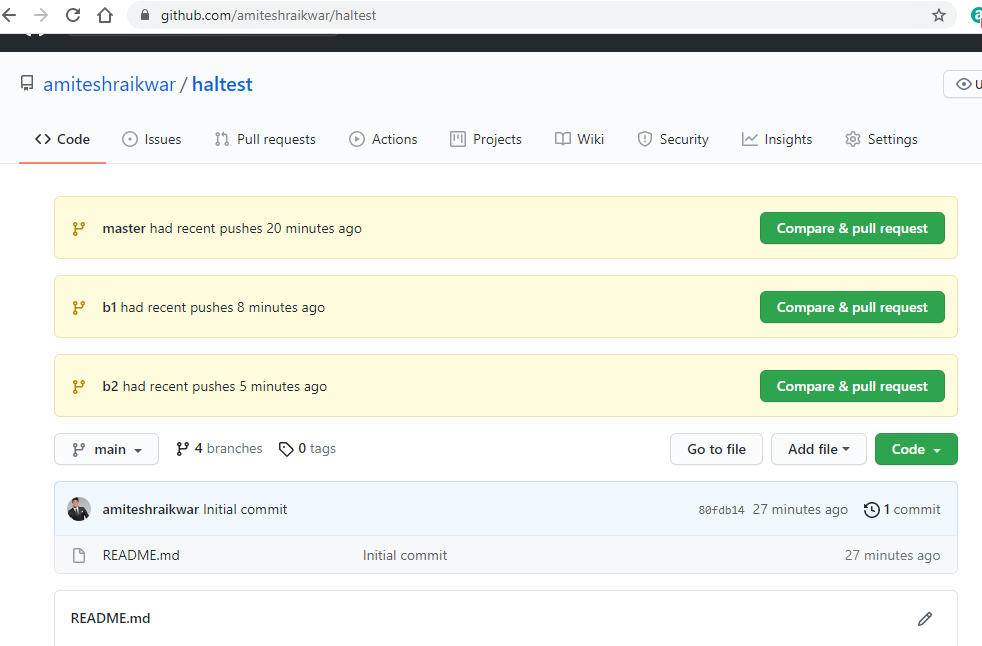
With the help of origin means our url is set, whenever we will run check we will see ourselves working on the origin as shown below. Origin will hold our url to allow not enter url again while pushing and pulling



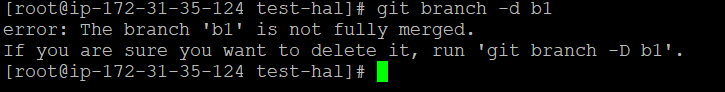


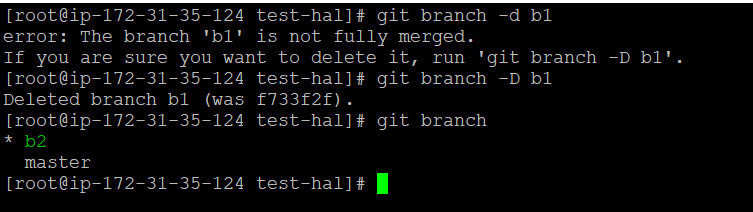


Now branch wise we have pushed the files to github



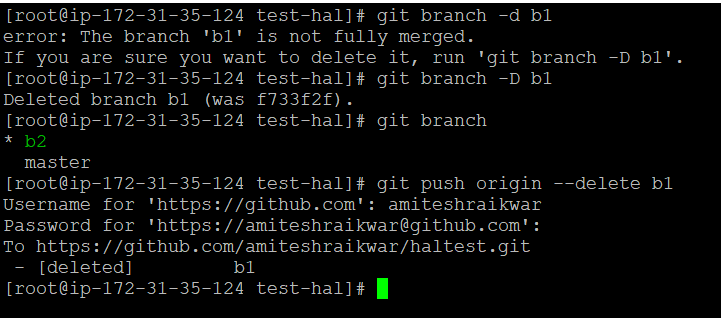
To delete the branch we will run git branch –d b1 , if branch is not merged than we need to run git branch –D b1





The b1 branch is deleted from local workstation repository not from github.

Now to delete it from github also we will run below command



To delete unmerged branch we use D & for merged branch d

