How to undo last commit in Git?

There will be different scenarios when you require undoing a commit in Git. Depending on the type, level of criticalness you may use different options/commands for reverting back the commit in Git.

In this tutorial, you may find different ways of undoing the previous commit.

Method 1: Using the reset command with hard and soft options

By definition, the **reset command** of Git is used to set the current HEAD to the specified state. So, you may use the reset command to revert back the last commit or back to the specified state. For example:

```
git reset --hard HEAD~1
```

This command will make the Git move the pointer of HEAD back to the previous commit. So, your last commit is undone and any files added or changes made are removed.
Keeping the changes in previous commit example

Basically, the reset command has three forms of invocation. The one is used in the above command i.e. **–hard**. The other two are:

- — mixed
- –soft

By using the **–soft flag**, you may keep the changes in the last commit while moving the HEAD pointer back to the last commit. So, for example, you do not want to discard changes made: files added, or any other changes. For that, you may use the –**soft** in reset command as used below:

> *$ git reset –soft HEAD~1*

If you run this command after the above command:

> *$ git status*

You will see, the changes in the last commit are preserved, however, the HEAD pointer is moved back to the previous position.

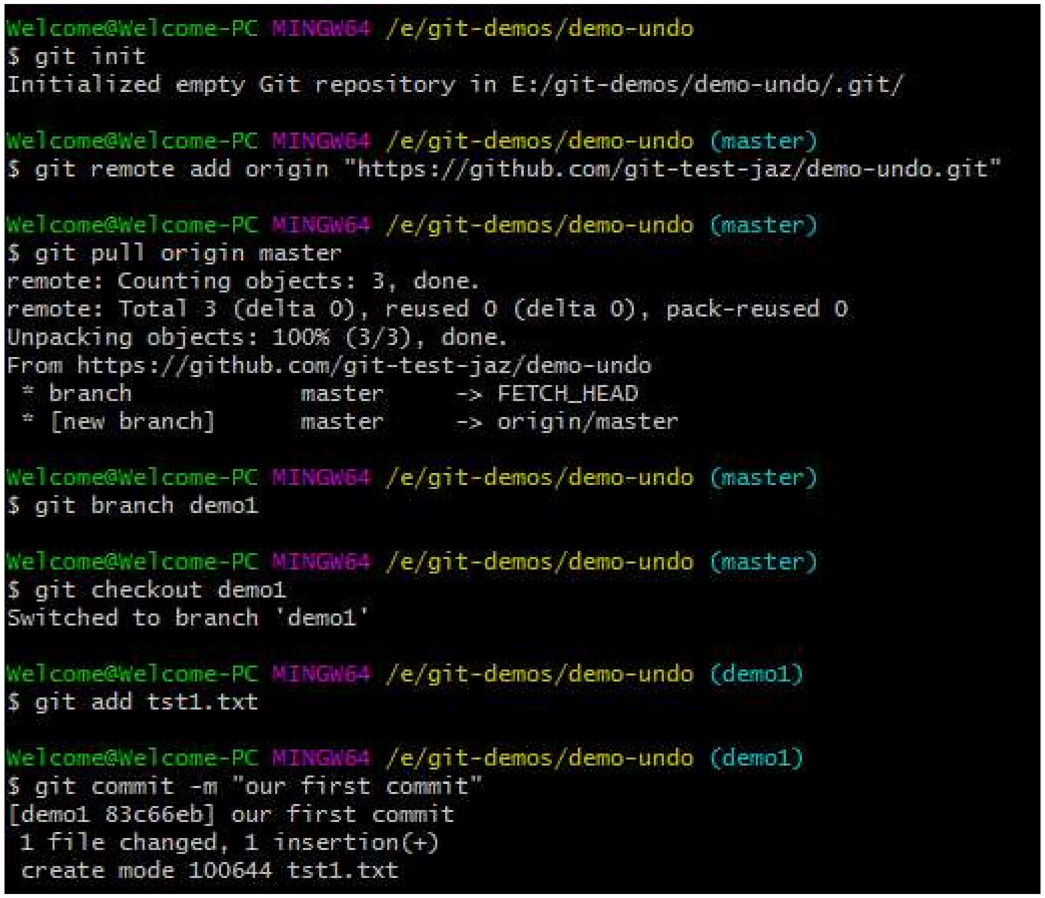
Still unclear, let us go through adding files in a branch, committing the changes and then doing hard and soft operations for undoing changes in the example below.

The example of undoing commit in Git

To understand the undo process by using Git restart with **–hard** and **–soft** flags, let us start with the basics.

For the demo, I have created a remote and local repository. The remote repository is created on Github and local on a Window system.

By right clicking on the local system folder, I opened the Git Bash (as explained in this tutorial). Fast forward, I ran a few commands as shown in the graphic below – for synchronizing the remote branch with local, creating a new branch locally and started adding files:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo
$ git init
Initialized empty Git repository in E:/git-demos/demo-undo/.git/

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (master)
$ git remote add origin "https://github.com/git-test-jaz/demo-undo.git"

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (master)
$ git pull origin master
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/git-test-jaz/demo-undo
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (master)
$ git branch demo1

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (master)
$ git checkout demo1
Switched to branch 'demo1'

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git add tst1.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git commit -m "our first commit"
[demo1 83c66eb] our first commit
 1 file changed, 1 insertion(+)
 create mode 100644 tst1.txt
```

Most probably, you already know about all that, this is just for beginners (others may skip to next heading below). So, quickly these commands are executed until our first commit:

> *$ git init*

(For initializing the empty git repository locally)

```
$ git remote add origin https://github.com/git-test-jaz/demo-undo.git
```

For synchronizing the remote repo with local.

```
$ git pull origin master
```

(downloading any content in the remote repository to the local)

```
$ git branch demo1
```

Creating the first local branch where we will commit three times and undo the committing to level 1 and 3 later.

```
$ git checkout demo1
```

This command is executed so we may add files in this branch.

After creating *tst1.txt* file in the local folder, execute the **add command** for adding a file in demo1 branch:

```
$ git add tst1.txt
```

Performing first commit:

```
$ git commit -m “our first commit”
```

The above graphic shows all these commands and output messages on Git Bash.

Pushing these changes to the remote repository:

```
$ git push origin demo1
```

If you refresh you remote repo, it should display the locally created branch and a text file with the commit message.

For learning how to undo commits in Git, I have created two more text files (tst2.txt and tst3.txt) on local repo and executed the commit commands as follows:

```
$ git add tst2.txt
```

Commit2 with a message:

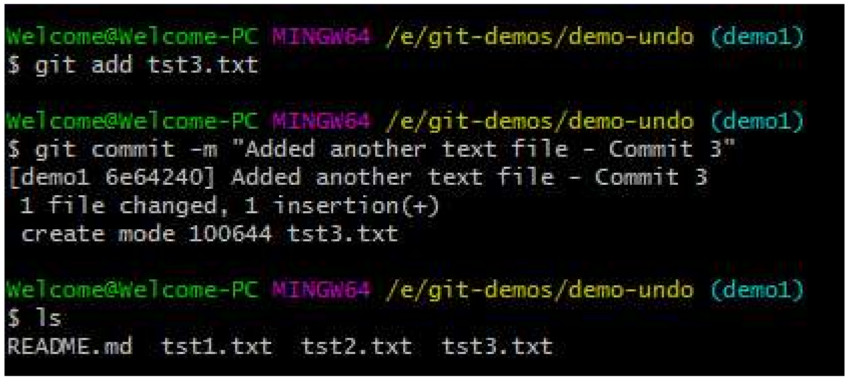
> *$ git commit -m "Added text file – Commit 2"*

Adding file 3:

> *$ git add tst3.txt*

Commit number 3 command:

> *$ git commit -m "Added another text file – Commit 3"*



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git add tst3.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git commit -m "Added another text file - Commit 3"
[demo1 6e64240] Added another text file - Commit 3
 1 file changed, 1 insertion(+)
 create mode 100644 tst3.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ ls
README.md  tst1.txt  tst2.txt  tst3.txt
```

You can see three files are listed as a result of running **$ ls** command after making this commit for the demo1 branch.

The files are: tst1.txt, tst2.txt, tst3.txt

Note: I ran commit command three times as I want to show you reverting back to the last commit as well as to initial level i.e. to the point where we had no any text file.

Running the reset command for undoing the last commit

Finally, we are reached to the point where I will execute the reset command.

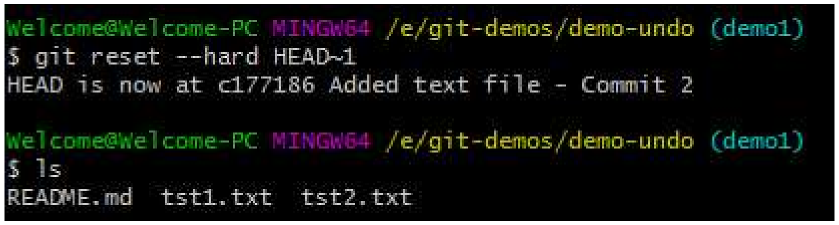
The first test is running the reset command with –hard flag. See how it outputs:

```
$ git reset --hard HEAD~1
```

This is followed by running this command:

> *$ ls*

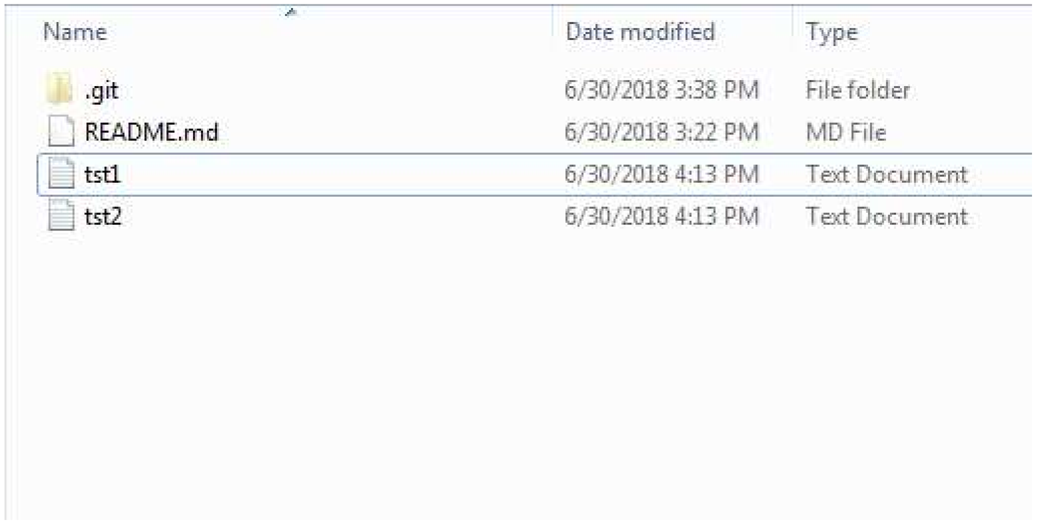
The output of these two commands:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git reset --hard HEAD~1
HEAD is now at c177186 Added text file - Commit 2

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ ls
README.md  tst1.txt  tst2.txt
```

There you go, the "HEAD is now at c177186 Added text file – Commit 2" i.e. to our second commit; so we have undone the last commit.

Running the **$ls** command also shows only two files now. If you go to the local repo folder, you will see the tst3.txt file is also removed.



Name	Date modified	Type
.git	6/30/2018 3:38 PM	File folder
README.md	6/30/2018 3:22 PM	MD File
tst1	6/30/2018 4:13 PM	Text Document
tst2	6/30/2018 4:13 PM	Text Document

This is because of the **–hard** flag with reset command. So, you have to be careful as reverting back to last commits. You may lose your important files if undone carelessly. I will show what happens if we used –soft flag.

Reverting back to first commit rather than last commit example

If you want to go back to first or any other commit rather than the last commit, you may provide a desired number with the HEAD i.e.

Instead of HEAD~1, use the HEAD~2 or HEAD~3 etc.

For the example, I am again adding the third file (tst3.txt), making the commit again and then execute with the HEAD~3 option as follows:

> *$ git add tst3.txt*

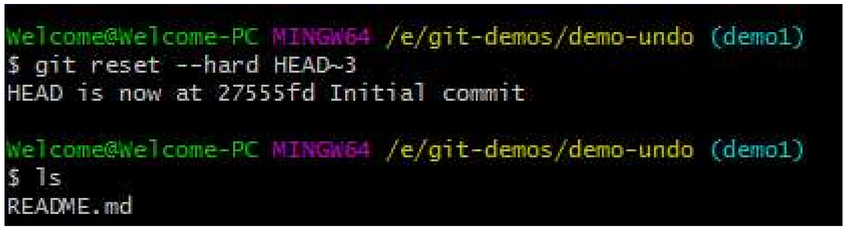
This is followed by:

> *$ git commit -m "Added tst3 again – commit 4"*

And finally executing the reset command:

> *$ git reset –hard HEAD~3*

The result is:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git reset --hard HEAD~3
HEAD is now at 27555fd Initial commit

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ ls
README.md
```

YES, you can see the $ ls command is listing only the README.md file – no text files that we committed by first, second and third commits.

Again, if you go to the local repo folder, it should only have the README.md file – all text files should have been removed.

Using the –soft flag for undoing last commit example

Now let us go through using the *–soft flag* in reset command for undoing the last commit. For that, I am again adding three files and performing the commits one by one for each change as in above section.

After that, we have three files in our demo1 branch in the local repository. The three files are also visible in the local repo folder.

Running the reset with –soft

Before running reset command, I executed the status command as follows:

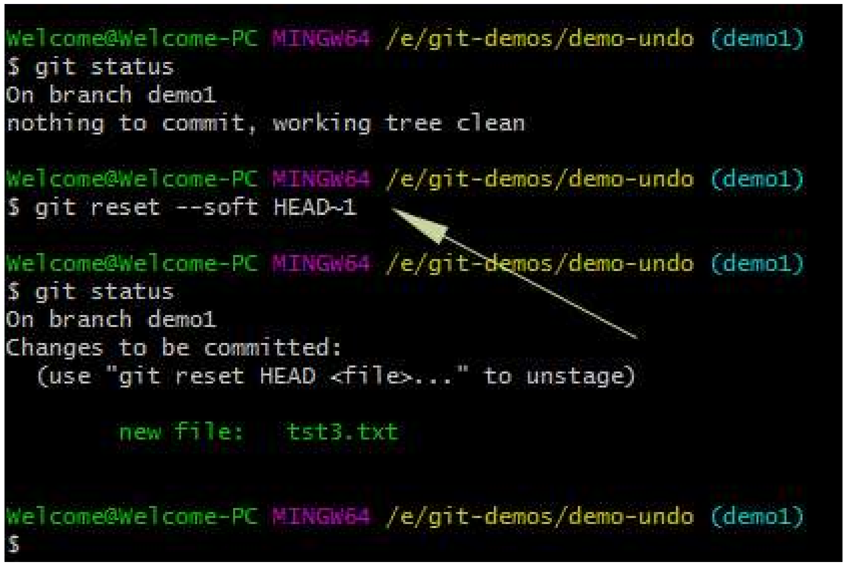
> *$ git status*

This is followed by reset with –soft:

> *$ git reset –soft HEAD~1*

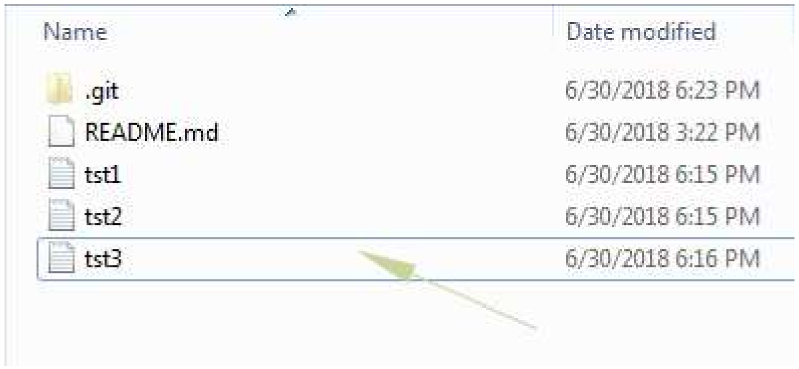
This command tells Git to move the pointer to the one step back without removing the changes– i.e. our second commit (before that, it was on commit 3).

I again ran the status command. See what are the output of all the commands on Git Bash:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git status
On branch demo1
nothing to commit, working tree clean

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git reset --soft HEAD~1

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git status
On branch demo1
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   tst3.txt


Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$
```

You can see, the second status command shows new file: tst3.txt as compared to the first command. Also, as I checked the local repo folder, it shows all files (tst1.txt, tst2.txt, tst3.txt) unlike the **reset with –hard** flag that showed only two files (tst3.txt was removed).



```
Name            Date modified
.git            6/30/2018 6:23 PM
README.md       6/30/2018 3:22 PM
tst1            6/30/2018 6:15 PM
tst2            6/30/2018 6:15 PM
tst3            6/30/2018 6:16 PM
```

Conclusion

On the basis of above experiment, we may conclude that if you require to undo the last commit when you are sure to discard everything associated with that commit, then use the reset command with –hard flag.

If you require keeping the changes (like holding the code files or any other changes) then use the reset with –soft flag.

We also saw, not only you may undo the last commit by reset command but may go to any level of commits i.e. earlier commits as well by specifying the number in HEAD~.

Method 2: Undo the last commit message example

As with every commit, a message is also saved. This is quite likely that spelling mistakes may occur as well as the message may be incomplete or more information is required to be added then the last commits' message.

In the case when you only require changing the message in the last commit, you should not use the reset command to go through all process again.

Instead, you may use easier option – **commit with –amend flag**.

Demonstration

In the following example, I have added a text file in the local branch. This is followed by using the commit command with a message. I also used push command with –force flag for submitting the changes in the remote repository.

In the message, I "accidentally" made typos and want to fix the message. This is how the message is fixed.

Add command:

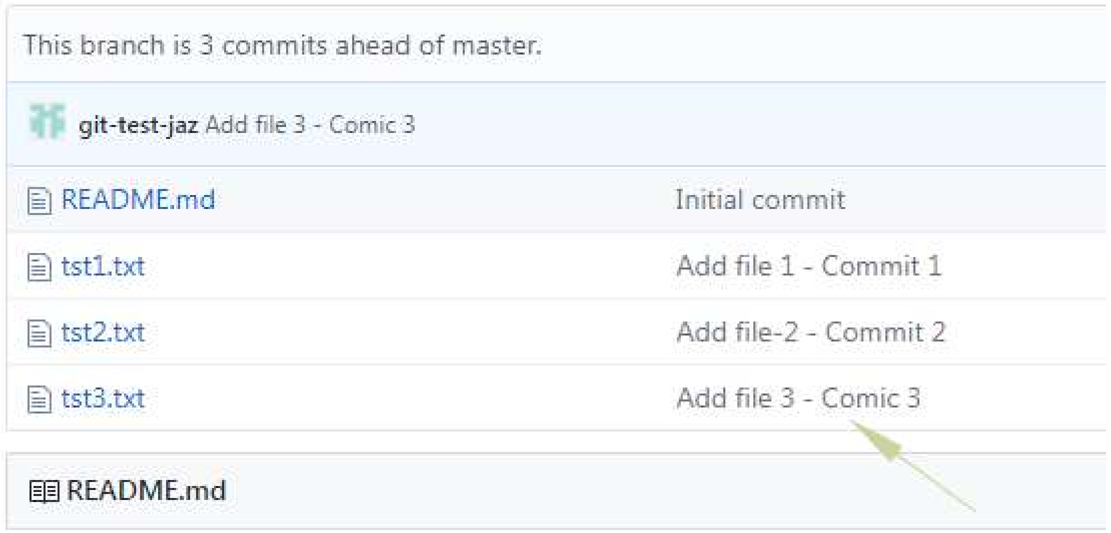
```
$ git add tst3.txt
```

Running the commit with the wrong message:

```
$ git commit -m "Add file 3 – Comic 3"
```

There, I actually wanted to use the word Commit rather *Comic*. However, I also ran the push command and changes were "live" on Github:

```
$ git push origin demo1 –force
```



This branch is 3 commits ahead of master.

git-test-jaz Add file 3 - Comic 3

README.md	Initial commit
tst1.txt	Add file 1 - Commit 1
tst2.txt	Add file-2 - Commit 2
tst3.txt	Add file 3 - Comic 3

README.md

Fixing the message:

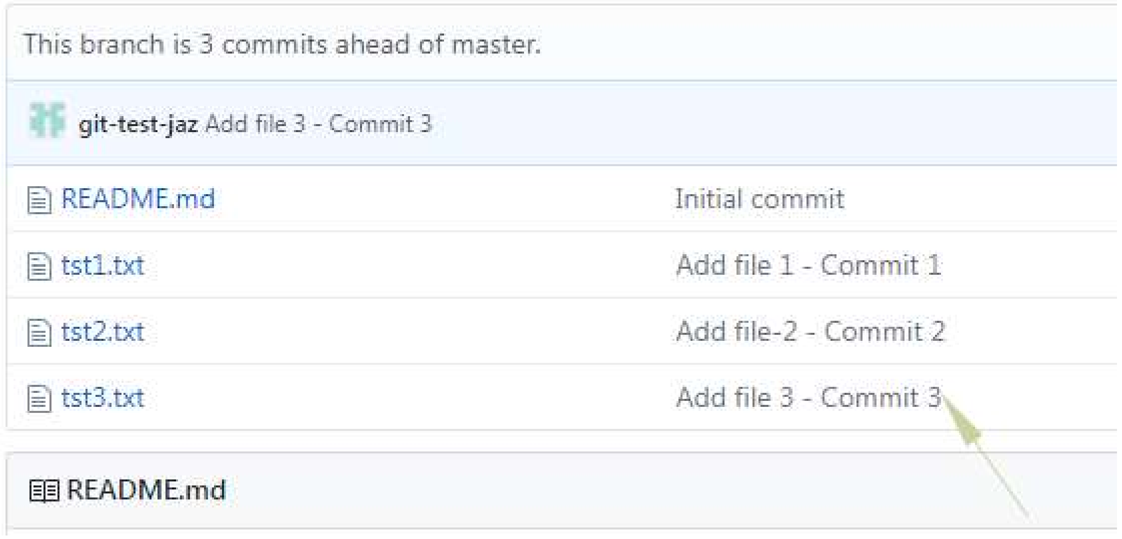
Now running the commit command again with **–amend flag:**

```
$ git commit –amend -m “Add file 3 – Commit 3”
```

Again executed the push command to fix the message in the online repo as well:

```
$ git push origin demo1 –force
```

The result in online repo:


This branch is 3 commits ahead of master.

git-test-jaz Add file 3 - Commit 3

README.md	Initial commit
tst1.txt	Add file 1 - Commit 1
tst2.txt	Add file-2 - Commit 2
tst3.txt	Add file 3 - Commit 3

README.md

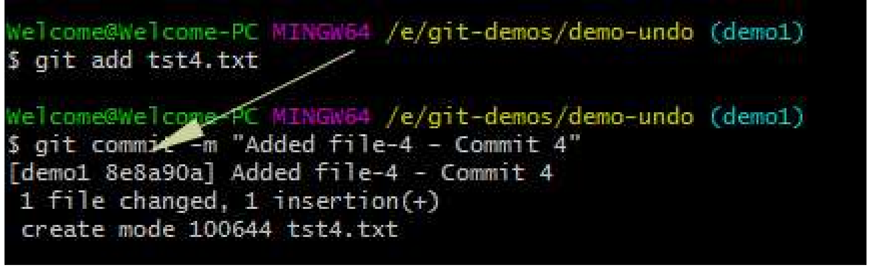
You can see, the amendments are made in the online repository without using the reset command.

Method 3: Using the revert command for undoing commit

Another way of undoing the commit (last or any) is using the revert command. In this way, you have to provide the commit id in the **revert** command.

You may give the shortcode of SHA1 or full code in the revert command. The following commands show reverting back the last commit and any other specified commit by **revert** command.

For the example, I have added another file (tst4.txt) file in our above-created branch and ran the commit command as shown below:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git add tst4.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git commit -m "Added file-4 - Commit 4"
[demo1 8e8a90a] Added file-4 - Commit 4
 1 file changed, 1 insertion(+)
 create mode 100644 tst4.txt
```

The arrow shows the commit id as we executed the commit command.

Now execute the revert command with that code, for example:

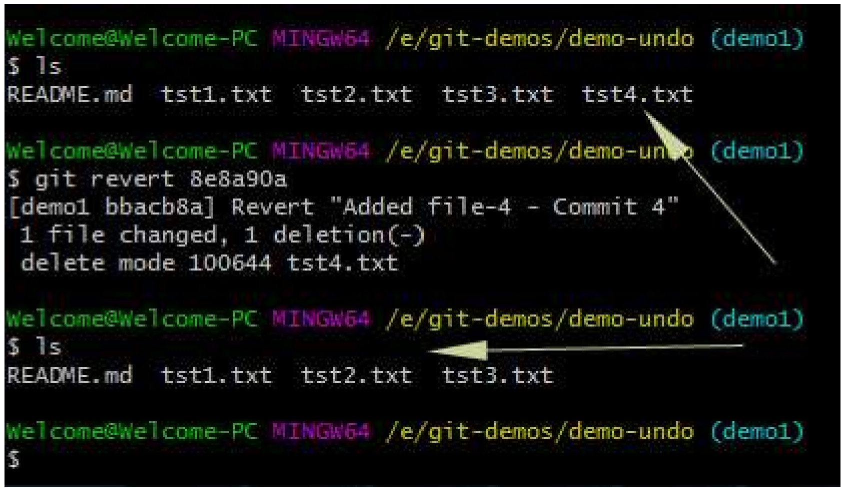
> *$ git revert 8e8a90a*

It will take to another terminal asking to enter the message. After entering a message, press the following keys (one by one):

> ***Esc : w q***

And press **Enter**.

The output of the test during this test is:



```
Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ ls
README.md  tst1.txt  tst2.txt  tst3.txt  tst4.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ git revert 8e8a90a
[demo1 bbacb8a] Revert "Added file-4 - Commit 4"
 1 file changed, 1 deletion(-)
 delete mode 100644 tst4.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$ ls
README.md  tst1.txt  tst2.txt  tst3.txt

Welcome@Welcome-PC MINGW64 /e/git-demos/demo-undo (demo1)
$
```

You can see, the $ ls commands show four files before running the revert command. After that, it is showing only three files, thus the commit of tst4.txt file is undone.

To get the SHA-1 code for the HEAD, you may run this command:

> *$ git rev-parse HEAD*

This should return full long SHA-1 that you may use in the revert command. Similarly, this command returns the shortcode of HEAD:

> *$ git rev-parse –short HEAD*

Or, simply use the following command:

> *$ git log*

That should return handy information about the commits. Get the code there for the required commit that you want to revert.