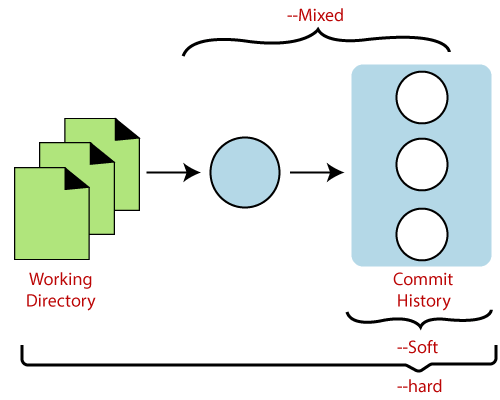
Git Reset

The term reset stands for undoing changes. The git reset command is used to reset the changes. The git reset command has three core forms of invocation. These forms are as follows.

* **Soft**
* **Mixed**
* **Hard**

If we say in terms of Git, then Git is a tool that resets the current state of HEAD to a specified state. It is a sophisticated and versatile tool for undoing changes. It acts as a **time machine for Git**. You can jump up and forth between the various commits. Each of these reset variations affects specific trees that git uses to handle your file in its content.

Additionally, git reset can operate on whole commits objects or at an individual file level. Each of these reset variations affects specific trees that git uses to handle your file and its contents.



Git uses an index (staging area), HEAD, and working directory for creating and reverting commits. If you have no idea about what is Head, trees, index, then do visit here [Git Index](https://www.javatpoint.com/git-index) and [Git Head](https://www.javatpoint.com/git-head).

The working directory lets you change the file, and you can stage into the index. The staging area enables you to select what you want to put into your next commit. A commit object is a cryptographically hashed version of the content. It has some Metadata and points which are used to switch on the previous commits.

Let's understand the different uses of the git reset command.

Git Reset Hard

It will first move the Head and update the index with the contents of the commits. It is the most direct, unsafe, and frequently used option. The --hard option changes the Commit History, and ref pointers are updated to the specified commit. Then, the Staging Index and Working Directory need to reset to match that of the specified commit. Any previously pending commits to the Staging Index and the Working Directory gets reset to match Commit Tree. It means any awaiting work will be lost.

Let's understand the --hard option with an example. Suppose I have added a new file to my existing repository. To add a new file to the repository, run the below command:

1. $ git add <file name>

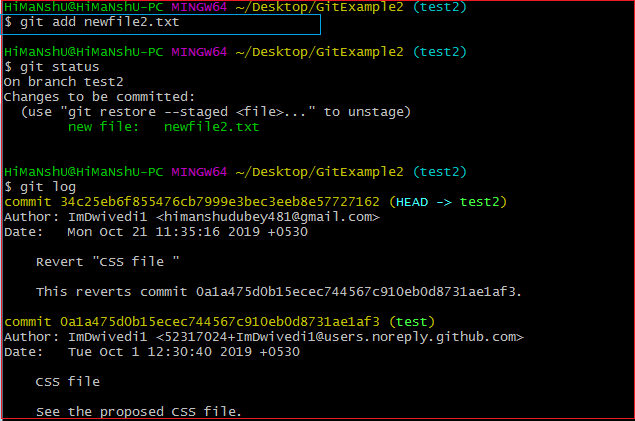
To check the status of the repository, run the below command:

1. $ git status

To check the status of the Head and previous commits, run the below command:

1. $ git log

Consider the below image:



In the above output, I have added a file named **newfile2.txt**. I have checked the status of the repository. We can see that the current head position yet not changed because I have not committed the changes. Now, I am going to perform the **reset --hard** option. The git reset hard command will be performed as:

1. $ git reset --hard

Consider the below output:

Git Reset

As you can see in the above output, the -hard option is operated on the available repository. This option will reset the changes and match the position of the Head before the last changes. It will remove the available changes from the staging area. Consider the below output:



The above output is displaying the status of the repository after the hard reset. We can see there is nothing to commit in my repository because all the changes removed by the reset hard option to match the status of the current Head with the previous one. So the file **newfile2.txt** has been removed from the repository.

There is a safer way to reset the changes with the help of [git stash](https://www.javatpoint.com/git-stash).

Generally, the reset hard mode performs below operations:

* It will move the HEAD pointer.
* It will update the staging Area with the content that the HEAD is pointing.
* It will update the working directory to match the Staging Area.

Git Reset Mixed

A mixed option is a default option of the git reset command. If we would not pass any argument, then the git reset command considered as **--mixed** as default option. A mixed option updates the ref pointers. The staging area also reset to the state of a specified commit. The undone changes transferred to the working directory. Let's understand it with an example.

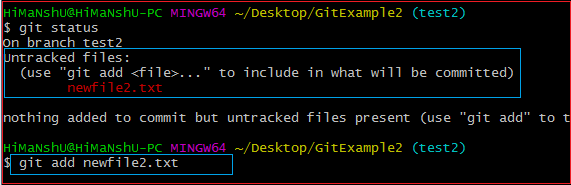
Let's create a new file say **newfile2.txt**. Check the status of the repository. To check the status of the repository, run the below command:

1. $ git status

It will display the untracked file from the staging area. Add it to the index. To add a file into stage index, run the git add command as:

1. $ git add <filename>

The above command will add the file to the staging index. Consider the below output:



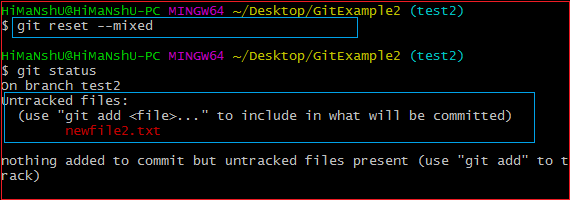
In the above output, I have added a **newfile2.txt** to my local repository. Now, we will perform the reset mixed command on this repository. It will operate as:

1. $ git reset --mixed

Or we can use only git reset command instead of this command.

1. $ git reset

The above command will reset the status of the Head, and it will not delete any data from the staging area to match the position of the Head. Consider the below output:



From the above output, we can see that we have reset the position of the Head by performing the git reset -mixed command. Also, we have checked the status of the repository. As we can see that the status of the repository has not been changed by this command. So it is clear that the mixed-mode does not clear any data from the staging area.

Generally, the reset mixed mode performs the below operations:

* It will move the HEAD pointer
* It will update the Staging Area with the content that the HEAD is pointing to.

It will not update the working directory as git hard mode does. It will only reset the index but not the working tree, then it generates the report of the files which have not been updated.

If -N is specified on the command line, then the statements will be considered as intent-to-add by Git.

Git Reset Head (Git Reset Soft)

The soft option does not touch the index file or working tree at all, but it resets the Head as all options do. When the soft mode runs, the refs pointers updated, and the resets stop there. It will act as git amend command. It is not an authoritative command. Sometimes developers considered it as a waste of time.

Generally, it is used to change the position of the Head. Let's understand how it will change the position of the Head. It will use as:

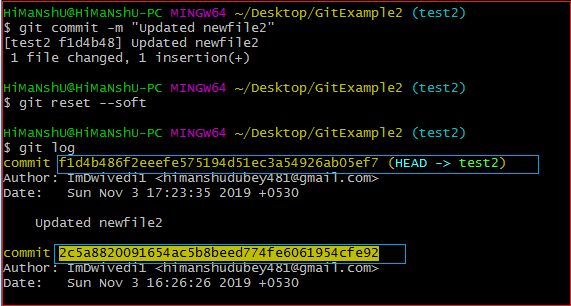
1. $ git reset--soft <commit-sha>

The above command will move the HEAD to the particular commit. Let's understand it with an example.

I have made changes in my file newfile2.txt and commit it. So, the current position of Head is shifted on the latest commit. To check the status of Head, run the below command:

1. $ git log

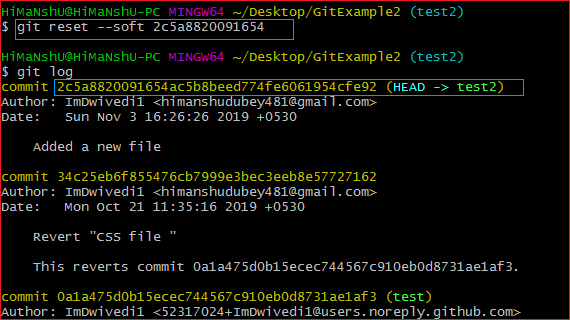
Consider the below output:



From the above output, you can see that the current position of the HEAD is on f1d4b486f2eeefe575194d51ec3a54926ab05ef7 commit. But, I want to switch it on my older commit 2c5a8820091654ac5b8beed774fe6061954cfe92. Since the commit-sha number is a unique number that is provided by sha algorithm. To switch the HEAD, run the below command:

1. $ git reset --soft  2c5a8820091654

The above command will shift my HEAD to a particular commit. Consider the below output:



As you can see from the above output, the HEAD has been shifted to a particular commit by git reset --soft mode.

Git Reset to Commit

Sometimes we need to reset a particular commit; Git allows us to do so. We can reset to a particular commit. To reset it, git reset command can be used with any option supported by reset command. It will take the default behavior of a particular command and reset the given commit. The syntax for resetting commit is given below:

1. $ git reset <option> <commit-sha>

These options can be

* --soft
* --mixed
* --Hard