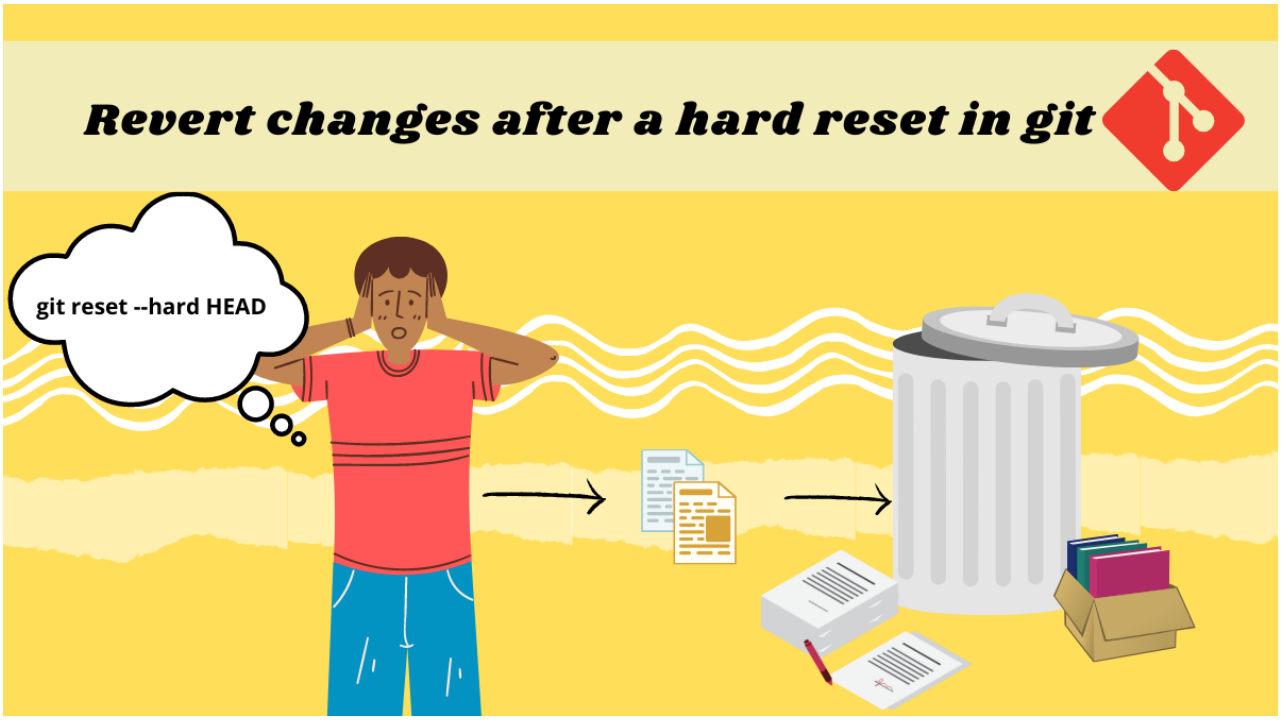
How to Recover a Deleted File in Git – Revert Changes After a Hard Reset

Zaira Hira



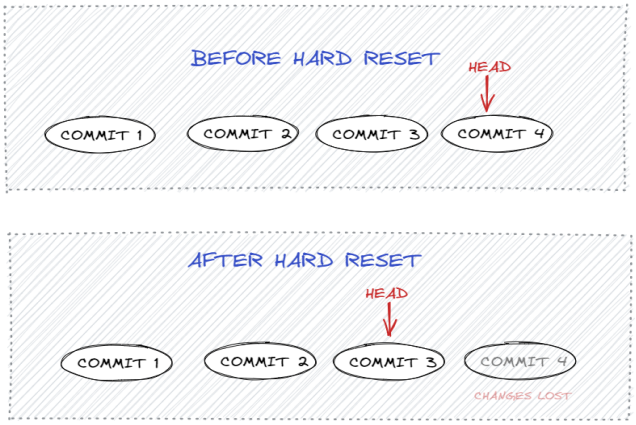
Git is a version control system that helps you keep track of the changes in a project's life cycle. It preserves the history of the project and allows you and your team members to coordinate effectively throughout.

There could be situations where you deleted a file and you want to recover it. The good news is that you can – most of the time – recover the files when using a version control system (VCS).

In this tutorial, we will learn the different methods that Git offers to restore deleted files.

How to Recover Files after Committing Changes

Let's say you committed a change but did a hard reset (`git reset --hard HEAD`) to a different commit which removed the latest commit from your current branch.



Hard reset explained.

In this case, you can restore the file using either `git checkout` or `git reflog`.

You can find the hash-ID of the previous commit from the command: `git log`.

After that, simply revert to the previous commit using:

```
git checkout <hash-id>
```

In case you don't have the hash ID, you can use the command `git reflog`.

`reflog` is a logging mechanism and keeps a track of all the changes against their unique `hash-id`.

Below is an example of the output of `git reflog`:

Output of `git reflog`

Pick the commit ID and use it to revert to that commit.

```
git reflog <hash-id>
```

How to Recover Files When Changes Are Staged but Not Committed

Suppose you staged a file with `git add <file-name>` and then did a hard reset with `git reset --hard HEAD` before committing. Afterward, you found out that the staged file is missing. In this case, also, you can recover the files.

We can use the command `git fsck` to recover the files after a hard reset.

What is `git fsck`?

`git fsck` stands for file system check. It checks for all the "dangling blobs" in the `.git` directory that are not part of any changes. For example, there could be some changes that were staged but not added anywhere.

Output of `git fsck`.

Once we are able to identify the "dangling blobs", we can view the details using `git show`.

```
git show f24facc98b387a375a50ba6d19193626cbfe7d45
```

Depending on the change, you'll be able to view your respective changes.

You might also want to save the changes in a file. You can simply redirect the output to a file using the `>` operator.

```
git show f24facc98b387a375a50ba6d19193626cbfe7d45 > restored_file.txt
```

Now, `restored_file.txt` will include the contents of the commit.

How to Restore Changes that Are Neither Committed nor Staged

In the case where the changes are neither staged nor committed, Git can't help you recover the files.

The reason is that the files weren't added to staging and Git can not tell the status of those files.

In this case, it would be helpful to search in temp files or the cached history of your text editor.

Wrapping up

When working on risky files, it is always better to use a VCS. In this way, the files will be preserved and the chances of accidental data loss are reduced.

In this tutorial, we learned how to restore deleted files whether they are staged or committed.

I hope you found this tutorial helpful. Thank you for reading till the end.

What's your favorite thing you learned from this tutorial? Let me know on Twitter!

You can also read my other posts here.
