**Git checkout:**

* Git checkout is the act of switching between different versions of a target entity.
* It operated upon three distinct entities such as files, commits and branches

**Checking out branches:**

* The git checkout command lets you navigate between the branches created by git branch.
* The git checkout command may occasionally be confused with git clone. The difference between the two commands is that clone works to fetch code from a remote repository, alternatively checkout works to switch between versions of code already on the local system.

**Commands:**

* **git checkout <commit ID>**

to set the HEAD to old commit ID

* **git checkout <branch name>**

to switch to another branch local branch

* **git checkout <remote branch> origin/<remote branch>**

to switch to remote branch

* **git checkout -b <new branch>**

this will create new branch and will switch to new branch.

* **Git checkout <file name>**
* **Git checkout . -🡪 for all the files**

If we want to discard the uncommitted changes from working directory. We can use checkout

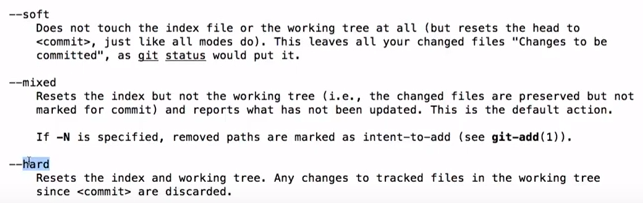
**Git Reset:**

With the reset, we can remove the recent commits. We cannot remove a specific commit with reset

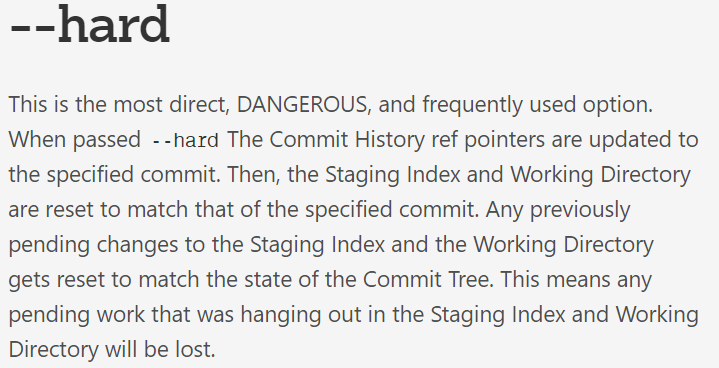
* **Git reset HEAD^ (it removes the last commit)**
* **Git reset HEAD~3 (it removes the last 3 commits)**
* **Git reset - -hard origin/master**
* **Git reset - -hard HEAD -> removes all uncommitted changes**
* Reset is used to remove the last made commits
* We should not use the commit which is shared to others added in repository
* Use this only if our commits are local
* It removes the commits from history, we can’t undo it
* It also untracks the file. We need to add the modified files after reset and commit freshly again

**Types of reset:**

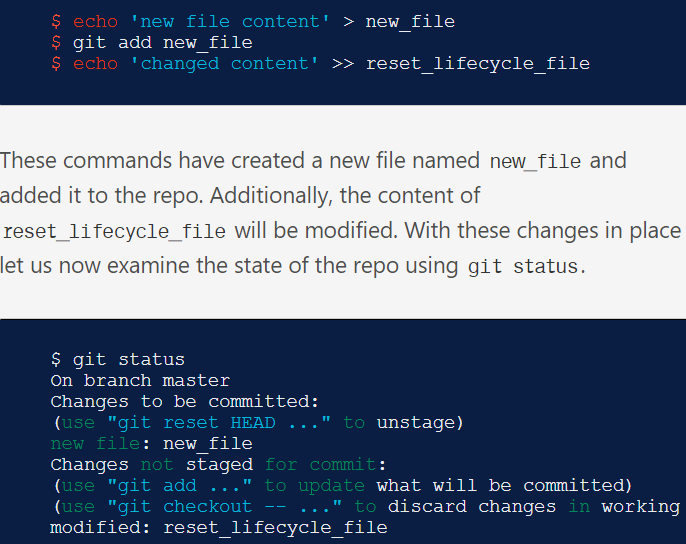
There is reset hard, soft and mixed types as below



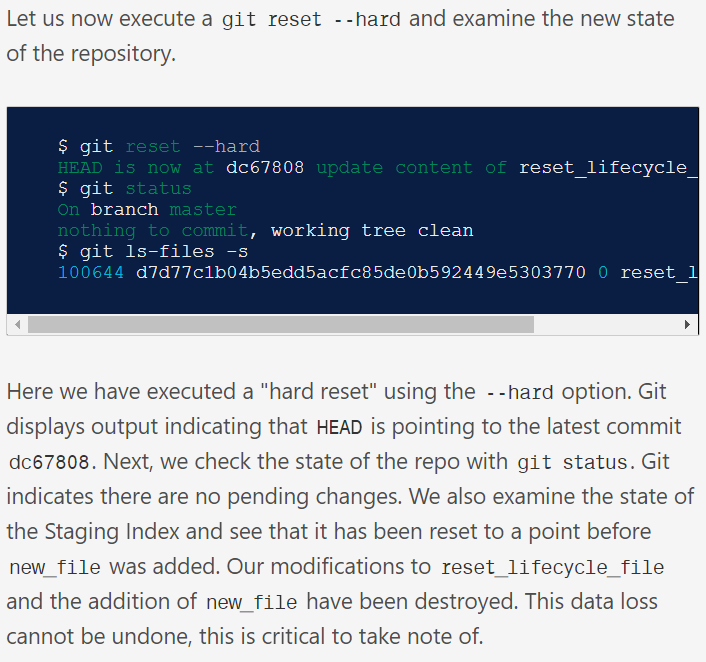
**Hard:**



**Example:**

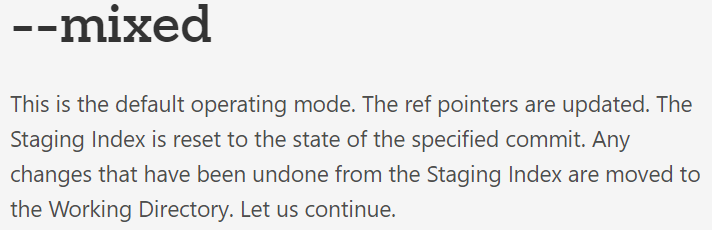


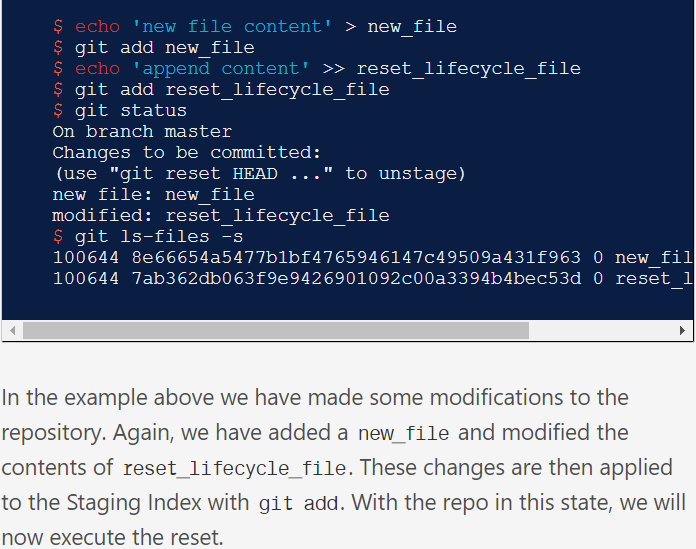
As above, two files are in staging which need to be committed.

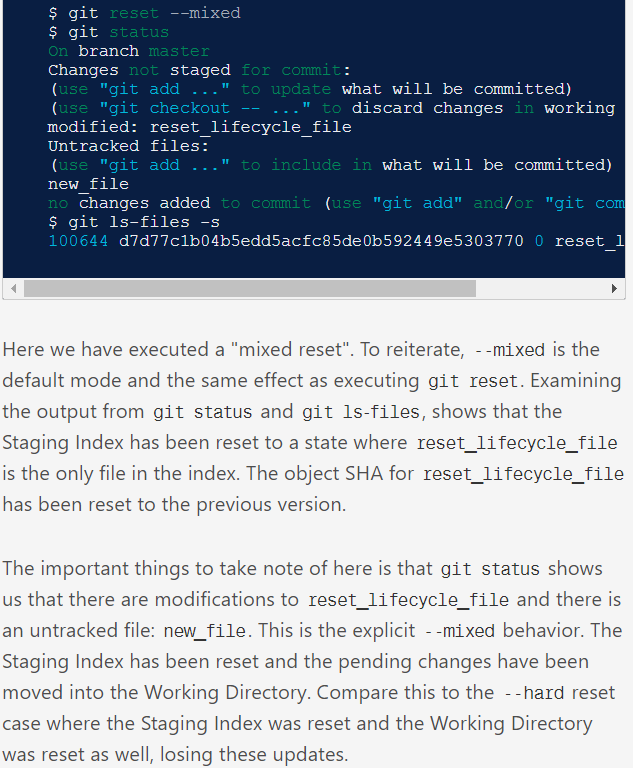


* Now, when we use reset with hard option. The staging data and unstaging data (working copy) both will be lost as above.

**Mixed:**

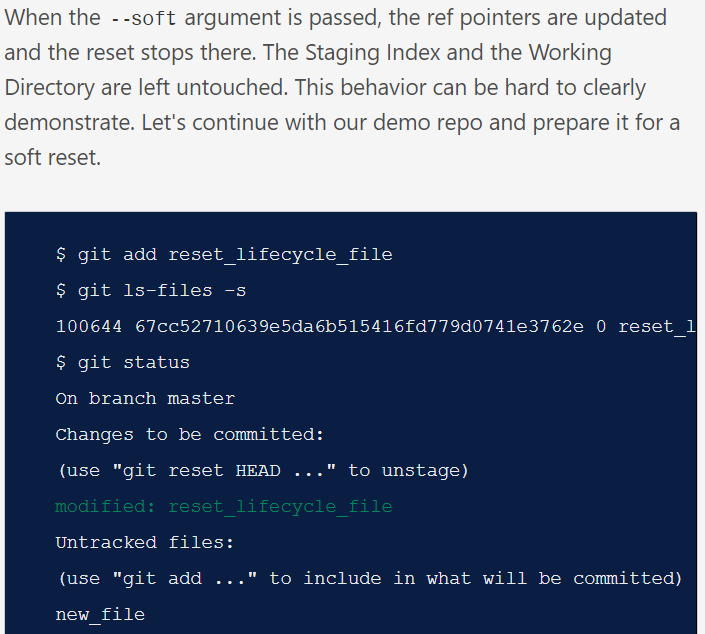


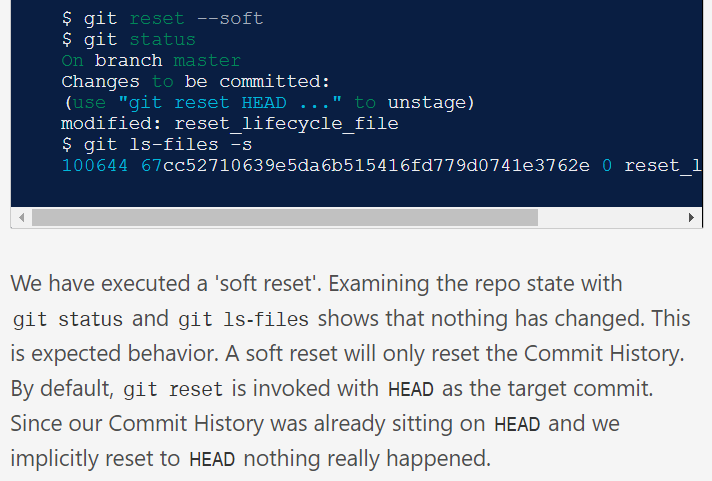




As above, with mixed type reset will not lose any data in staging. If there is any new file, it will reset to working copy. We need to add it and commit again.

**Soft:**





* As above, it will not touch anything in staging or working copy. It just resets the commit to head.
* By default, it is mixed type

**Git reset is used to remove the commit not to delete the data in a particular commit. I think it will be very useful when we want to edit the commit**

**With –hard, it removed the data with the commit also**

**Git revert:**

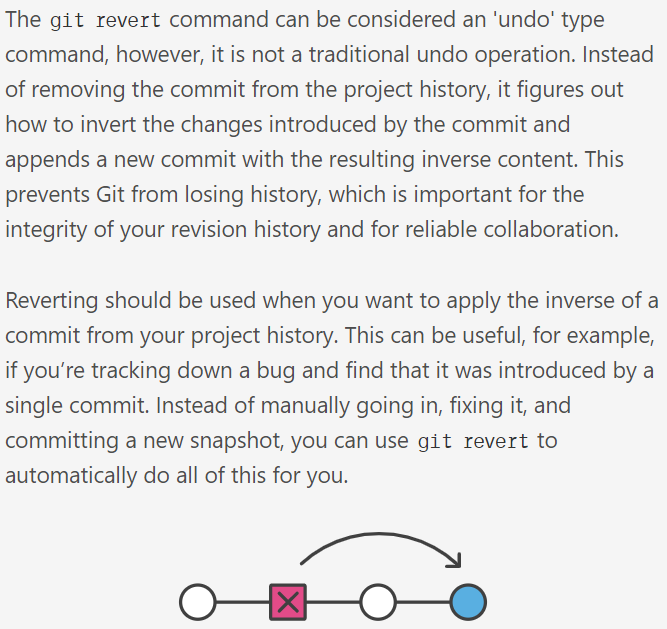
* When we want to remove the commit. With the reset, it may get conflicts while pushing to central repo. We can use revert instead

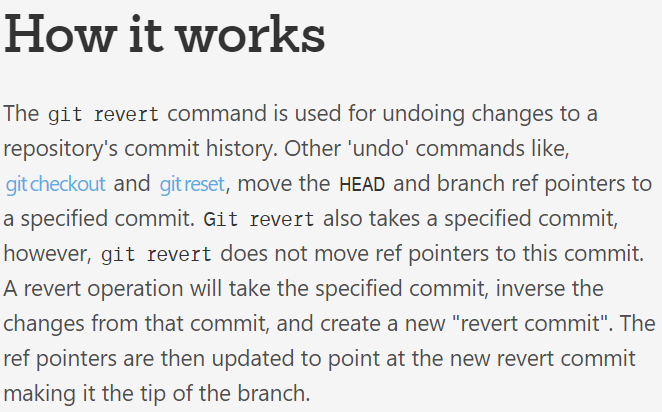
It undoes the changes and creates a new commit with new changes

* **Git revert <commit id>**

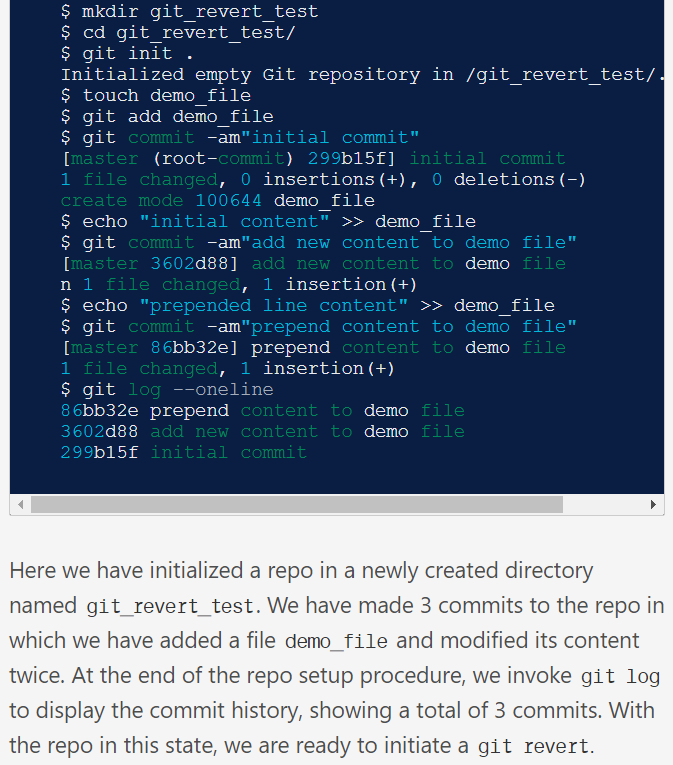
It removes the commit makes another commit for undo. We can see that details with git log

After this, we can push it to central repo without any issues, but with reset, it won’t happen

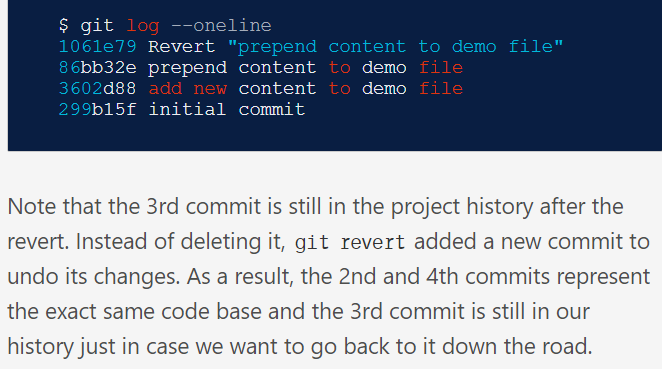




**Example:**







**Checkout vs Reset vs Revert:**

