Undoing things in Git
git reset & git revert

- by a practical example -

Let‘s start with a repository which has the following commits:

```
* 2a6fddf (HEAD -> master, origin/master) Second commit
* d2e4a38 Initial commit
```

We create a file „error.txt“, stage it and commit it:

```
touch error.txt
git add error.txt (or: git add .) [STAGING]
git commit -m „Error commit“ [COMITTING / ADDING TO HISTORY]
```

Afterwards we get the following history:

```
* 969b2bb (HEAD -> master) Error commit
* 2a6fddf (origin/master) Second commit
* d2e4a38 Initial commit
```

Now when testing our application we realize:
file error.txt does not work as expected. It produces errors.

We want to get rid of it again.

We have two basic options to undo our last commit:

1) Delete the changes in the commit completely (i.e.: delete the commit together with file „error.txt“)
2) Undo the commit, but keep the changes for fixing / updating them

First we apply option 1: Completely erase the commit and it‘s changes
(We will deal with option 2 later)

We do not have the option to delete a commit directly, like `git delete 969b2bb`.

Instead we tell git to move „back in time“.

That is exactly what git reset is doing.

Git reset

We now apply git reset to go back in time.

We want to go back to the commit BEFORE out last commit.

So we want to get back to commit `2a6fddf`:

```
* 969b2bb (HEAD -> master) Error commit
* 2a6fddf (origin/master) Second commit => this is were we want to be
* d2e4a38 Initial commit
```

We do this by the command:

```
git reset --hard  2a6fddf
```

And that‘s it! Now our history looks like this again:

```
* 2a6fddf (HEAD -> master, origin/master) Second commit
* d2e4a38 Initial commit
```

You see. The last commit is gone.

And same for our file „error.txt“. You can check this by running „ls“ in your directory. The file is not there anymore.

We can do the very same with the following „shortcut“ command:

```
git reset --hard HEAD^
```

This will tell git: Move back exactly ONE commit / undo the very last commit.

In case you want to go back multiple commits you can use the following command:

```
git reset --hard HEAD~<number-of-commits>
```

Some examples:

```
git reset --hard HEAD^ => goes back one commit
git reset --hard HEAD~1 => does the same as the command above!
git reset --hard HEAD~3 => undo the last 3 commits
```

Now let‘s handle option number 2: **Undo a commit, but keep the changes and re-consider them**

Let‘s assume, we are here again:

```
* 969b2bb (HEAD -> master) Error commit
* 2a6fddf (origin/master) Second commit => this is were we want to be
* d2e4a38 Initial commit
```

So we are at commit **969b2bb** and want to go back to **2a6fddf**

But this time we want to **keep** the file error.txt and fix the errors inside.

We do this by using git reset without the hard option.

So let‘s do:

```
git reset HEAD^1
```

Now we have successfully undone the last commit. And are here:

```
* 2a6fddf (HEAD -> master, origin/master) Second commit
* d2e4a38 Initial commit
```

But when we now run

```
git status
```

we will get the following output:

```
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

     error.txt

nothing added to commit but untracked files present (use "git add" to
track)
```

So yes, git has undone the last commit successfully. But the changes we have done in this commit are still in our directory, as you can see by the output of git status.

We can check this by running „ls“. We see that the file „error.txt“ is still there.

We can now fix the error in this file and commit it again. This way we can keep most of our work.

So in summary:

`Git reset –hard HEAD^`
=> Put the changes of your last commit to the trash because you do not want them anymore / restart

`Git reset HEAD^`
=> Undo the last commit. But keep your changes you have done in this commit → for fixing / update / tweak them. After fixing them, you can commit them again (this time without errors).

Undoing an already pushed commit

Some times we do not realize our errors. Until we push them. And others realize them and tell us about it :-)

We need to find a way to fix the error commits that are now already shared accross our team!

This time we cannot just go back in time. Git reset ist just for undoing things locally.

For already pushed changes we need our last weapon:

`git revert`

Git revert will create a so called „inverted commit“. This is a new commit which will undo all changes from the previous commit.

So let‘s use the following scenario:

```
* 969b2bb (HEAD -> master, origin/master) Error commit
* 2a6fddf Second commit => this is were we want to be
* d2e4a38 Initial commit
```

We see we already pushed our error commit **`969b2bb`** to the central master (origin/master). Damn it.

We want to go back to the world how it was at commit: **`2a6fddf`**

But we cannot go back there directly.

We now have to create a commit which undos everything that happened in our last commit **`969b2bb`**.

We do this by the command:

```
git revert 969b2bb
```

A dialog will open to specifiy a commit message for the revert commit (save the file to confirm):

Now the have the following state:

```
* 7d90bb2 (HEAD -> master) Revert "Error commit"
* 969b2bb (origin/master) Error commit
* 2a6fddf Second commit => this is were we want to be
* d2e4a38 Initial commit
```

Now we have successfully undone our pushed commit.

So if we now do „ls“ in our directory we see: The file error.txt is gone.

We now need to push the change to the others. So they will receive the fixing of the error too:

```
git push
```

Afterwards we have the state:

```
* 7d90bb2 (HEAD -> master, origin/master) Revert "Error commit"
* 969b2bb Error commit
* 2a6fddf Second commit => this is were we want to be
* d2e4a38 Initial commit
```

And now we are clean again!

Now with these methods you can feel safer with your produced errors. Everything can be undo.

Happy undoing!