# Git & Github Guide

## Reverting changes

There are many ways to actually reverting the changes that you've introduced in your git repository

**git checkout <log name>:**

The git checkout command operates upon three different entities which are **files, commits,** and **branches.** Sometimes this command can be dangerous because there is no undo option available on this command.

We could perform many operations by git checkout command like:

- to switch to a specific **branch**
- create a new **branch**
- checkout a remote **branch**

The **git branch** and **git checkout** commands can be integrated.
Edits the files in the current commit as in they were in the <log name> commit

**git revert <log name>: related**

-- Makes a new commit with the changes of the <log name> one. --

Given one or more existing commits, revert the changes that the r*elated patches* introduce, and record some new commits that record them.

*This requires your working tree to be **clean** (no modifications from the HEAD commit).*

**git reset <log name>:**

-- Resets current **HEAD** to the specified state. It mainly allows the user to un-stage files or changes to the **HEAD** commit. --

It deletes previous commits leaving you only with the **<log name>** specified one, It remains the changes made in previous commits, **they're just not staged yet** it is equivalent to **--mixed**

***(You could use it, for example, to group commits into one)***

When no arguments are given, It will just un-stage all files back to the working directory:

- **with** --soft **flag, it stages those changes.** This is mainly used to delete a commit log without removing its changes.
- **with** --hard **flag, it deletes every previous change made after the <log name>.** Deletes, every single change after the specified commit.

## Shortcuts

**git reset HEAD@{1}**

Resets to the last commit made on the project.

**git reflog <logname>**

Shows all the snapshots where you've made changes, and in order to revert your working tree to said snapshot, you could make a **checkout** to the **log-name** that belongs to the snapshot in question, even after dangerous operations like a **git reset --hard**. pic related

Said checkout doesn't last long in the garbage collector, you could make a branch to later merge it.

**(Ver si haciendo un commit con dicho estado tendrá el mismo efecto)**

## Managing branches:

-- For adding ***features*** to your code-base without making important changes to it, you could create different features that go in parallel with your workflow. --

**git checkout** *-b* **<branchname> || git branch <branchname>:**

-- *Switch branches or restore working tree files* --

The git checkout -b option is a convenience flag that performs run git branch **<new-branch>** operation before running git checkout **<new-branch>**.

The second one only creates the new said branch. Using this command you could also check previous states of the project, using **git checkout <log-name>**, and then to attach the HEAD to the actual state of the project, you only need to **checkout** to the **main** or *said* branch.

**git branch <branchname>**

With no parameters it shows all the local branches, adding the "**-a**" flag will make sure remote branches are also included in the list. Including a **<branch name>** it will create a new branch. Branches are created from the main HEAD.

**git branch -D <branchname>: || git branch -d <branchname>:**

It deletes the mentioned branch, if the letter is in minus it only works if those changes were previously merged into master.

**git branch -m <old> <new>:**

Changes the name of said branch

**git branch --merged:**

Shows the branches that had been merged.

**git checkout <branch name>:**

Like said previously it changes to the mentioned branch

**git merge <branch name>:**
Merges the <branch name> branch to the current one

If the branch that we're going to introduce has a set of changes that differ too much from the original one or has changes that interfere with the codebase or changes made in another branch. a **Conflict** will occur. In case that we create a branch that

it's just ahead of the master branch updates, we could merge it with no problems. In case that we have to merge a branch that it's not up to date with the master branch changes, It would be a 'recursive merge' and we have to write a commit message.

## git stash

This is used for saving not staged changes and new files **before** moving to a different branch or snapshot. Is commonly used when you don't want to fully commit the changes that you've made in the repository, but still want to save them somewhere.

The regular **git stash** command is used to temporarily save said changes, and to save them on a queue called stash list, that we could access using the **git stash list** option, this will show said list.

```
stash@{0}: WIP on design: f2c0c72...
stash@{1}: WIP on design: f2c0c72...
stash@{2}: WIP on design: eb65635...
stash@{3}: WIP on design: eb65635...
```

We could return to the last stash using the **git stash apply** command or the **git stash apply** `stash@{x}.`

We could also name said stashes using **git stash push -m "message".**

You can even turn a stash into a branch using **git stash branch** <branchname> [<stash>]

## Interactuando con Github:

**git remote add origin <repo link>:**

Just adds the origin alias to the main repository for us to push with:

(**git push origin master) Para subir todas las ramas,** podemos añadir el flag **--all** al push.

About pull requests...

## Modify commits

**git commit --amend -m:**
Edita los mensajes del commit en el cual te encuentras:
(revisa a ver si puedes hacer primero un git checkout <log> para editar commits anteriores)

**git log --stat:**
Muestra los commits con los  cambios que se realizaron entre ellos

**git log --oneline:**
Muestra los commits de manera corta
(revisa una atajo para esto)

**git cherry-pick <commit hash>:**
Copia el commit **<commit hash>** a la rama en la que se encuentra en el momento

**git commit -a:**
Sin necesidad de hacerle stage a los cambios hechos realiza un commit.

**git clean <repo link>:**
Deletes untracked elements

-d: Los directorios
-f: Los archivos