# Exercise block 2: Git as version control system – from commit history to undoing changes on different levels

## 8. History of your project

So far, we have used Git straight forward and did not look back, but of course, the purpose of a version control system is to be able to go back to old versions as well. Therefore, we now want to look at your commit history.

8.1. The command to get your commit history is `git log`. What kind of information do you get from it?
8.2. Find some further options to limit the output (e.g. only show commits since a specific date, a certain number of commits…)
8.3. How can you limit the history output to only the commits in which we changed something in the *randomNumber.py* file, with the oldest on top?

## 9. Jump back and forth between versions

9.1. Open the randomNumber.py script again and replace the "my" with "your" in the line, which writes to the log file. Close the python script.
9.2. Look again at your commit history, copy the SHA1 of the last commit, and then run "`git diff <SHA1>`". What do you see?
9.3. Add and commit your changes with commit message "replaced my with your".
9.4. In case you need to go back to a specific commit you can do so by running "`git reset <SHA1>`" which takes you to a specific commit or "`git reset HEAD~1`" which moves you back in your history by 1 commit. Try one of these commands and see what happens.
9.5. If you run `git log` again, it looks like you have lost the last commit, because you do not see the "replaced my with your" commit anymore. However, of course, it is not lost to Git. Type `git reflog` (can take some time) and try to understand the output of the command.
9.6. Copy the SHA1 (it is short, but still works) of the "replaced my with your" commit (from 9.3) and run "`git reset <SHA1>`" to reset your index to the most recent commit again. Attention: The SHA1 we are looking for does not have to be the top one, orient yourself by looking at your commit messages (useful commit messages make life easier).
9.7. You should be back at the latest version. If you use the `git reflog` command again, it will show you the way you have gone.
9.8. Make sure your working directory is clean. Running `git status` should output:

```
On branch master
nothing to commit, working tree clean
```

**10. Undoing things**

10.1. First, run `git log` and copy the SHA1 of the newest commit into an extra file named SHA1.txt and save this file in the folder *gitTutorial* (parent directory of *myFirstGit* directory). Probably you will not need this file. Nevertheless, it makes it easier if you mess up your Git repo completely, to come back to a state, from where you can continue the exercise session today.

10.2. Then, choose a SHA1 from any commit you like in the `git log` output and reset again, but this time, provide the `--soft` switch for the reset command.

```
git reset --soft <SHA1>
```

How does it influence the behavior of the reset?

Note: If you run `git reset` without further options like in 9.4 , it will use `--mixed` for its default behavior, instead of `--soft`

10.3. Come back to the "replaced my with your" commit as you did in 9.6

10.4. Now just go one commit back, by using the SHA1 of the second to latest commit but provide the `--hard` switch for the reset command this time. How does it influence the behavior of the reset now?

10.5. Don't go back to the newest commit, instead stay on this second to latest commit and open the *randomNumber.py* script again and replace the "my" with "our" in the line, which writes to the log file. Add and commit your changes with commit message "replaced my with our".

10.6. Have a look on the `git log` output. You should have successfully changed your commit history. You have undone the "my -> your" change and could continue with the "my -> our" change instead

Note: Using `git reset` is a reasonable way to undo changes and is used frequently, but it should generally be considered as a local method only because it changes your commit history. If you collaborate with others, you can probably imagine that it is not a good idea to remove commits from history if they are already public to others.

**11. Undo uncommitted changes**

So far, we have seen how to come back to old versions when everything is already committed. Let us have a look on situations where you want to undo changes, which are not committed yet.

11.1. Open the *randomNumber.py* again and add the following line in the main function:

```
print("debug print")
```

11.2. Close the script and add the change (but do not commit yet)

11.3. Of course, debug prints are generally not a reasonably way to debug your code. Sometimes we use them anyway, but then we do not want anyone to know, but

stupidly, we have already added the change. Run `git status`, it will tell you what you can do to unstage your change. Run the suggested command.

11.4. Run `git status` again, it will tell you what you can do to discard your change, run the suggested command.

11.5. What do the two commands do in terms of the 3 -area concept?

**12. Change commits**

12.1. Open the *randomNumber.py* again and change the name of the function
`get_color_by_dice`
to
`get_color_by_dice_roll`
but forget(!) to change the name in the main where it is called for now (a common mistake).

12.2. Close your script, add and commit your changes.

12.3. Now you realize that you have forgotten to change the name of this function in the main as well. Change the name so that the script runs smoothly again. Now, you could add and commit this change as a new commit as well, but of course, it makes more sense to have these two changes in one commit. To do this, first run
´ `git add randomNumber.py`
and afterwards
`git commit --amend`
After executing the commands, this file should show up:

```
changed functionname

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue May 28 23:32:08 2019 +0200
#
# On branch master
# Changes to be committed:
#       modified:   randomNumber.py
```

You can edit the commit message if you want and save it afterwards or just save it without changing it (type ":wq" to save and close the file).

The changes from the last commit and the changes afterwards should now be contained in one commit. Check the output of `git log` to see that only one commit for this exists.

**13. Keep your history** *(extra-task, if you get here in time)*

In exercise 10, we have already seen how to reset to an old version, change things and continue with new commits. `git reset` should generally be considered a 'local' undo method. You can easily mess up your repo if you change your commit history and your collaborators have still the old commit history. However, there is a second option to undo things: `git revert`

Instead of resetting and rewriting your history, `git revert` will create a new commit. The new commit will have the opposite action to the commit we want to undo.

13.1. Open the *randomNumber.py* again and add an arbitrary name in the line of your team members. Close your file, add and commit your changes.
13.2. Now use `git revert <SHA1 of the last commit>` to revert this change. Type :q to close the automatically opened file
13.3. Have a look to `git log` afterwards to see how Git tracks these changes in your commit history