Git Exercises

1. I create a local clone of the repository with the simple command:

   git clone https://github.com/BI-DS/GRA-4152

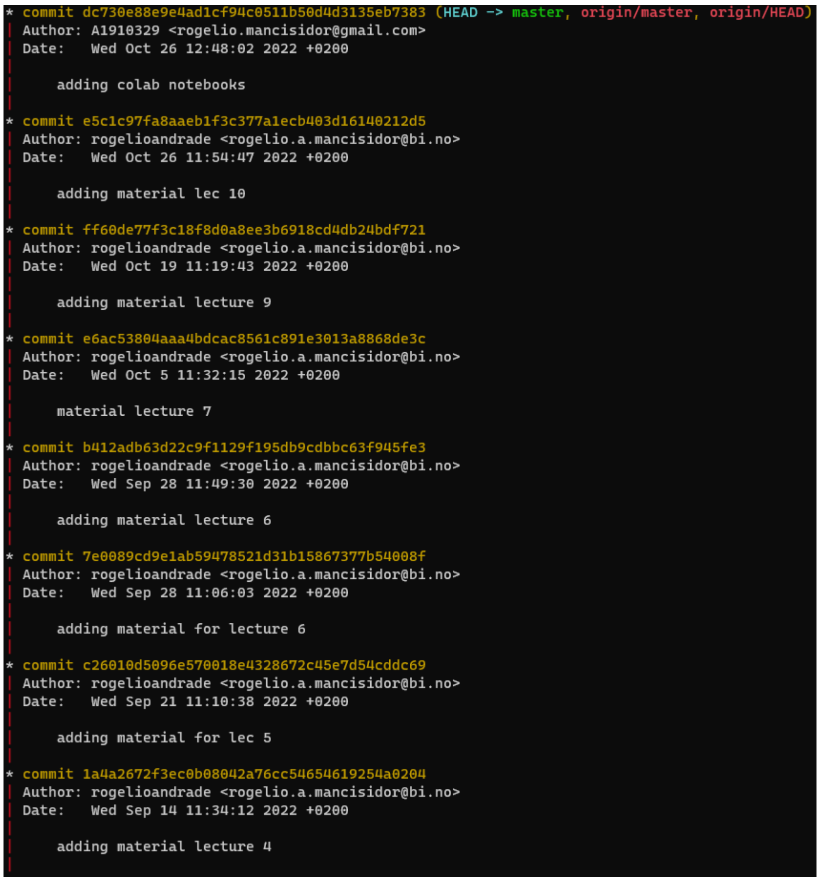
a) The simplest way to get a (text based) graphical representation of the version history is with the following command line:

   git log --all --graph

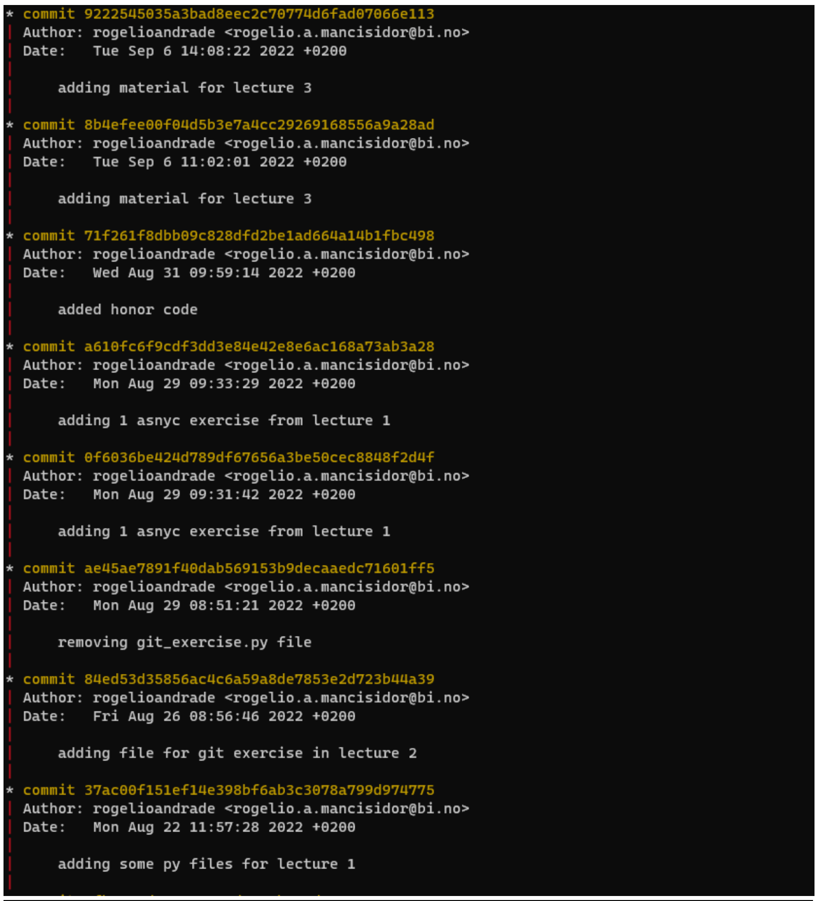
   This must be done after we have moved into the cloned repository with:

   cd GRA-4152

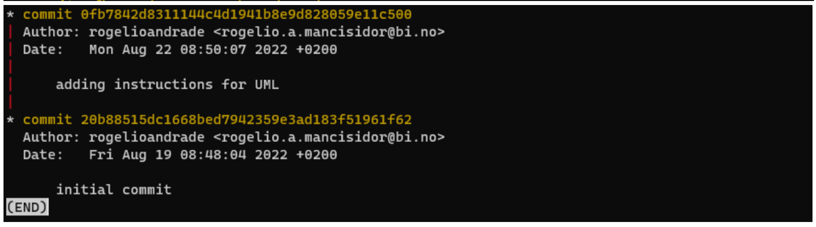
“--all" is to make sure all references are included, and “--graph" draws a text-based representation of commits. The result is:



```
* commit dc730e88e9e4ad1cf94c0511b50d4d3135eb7383 (HEAD -> master, origin/master, origin/HEAD)
| Author: A1910329 <rogelio.mancisidor@gmail.com>
| Date:   Wed Oct 26 12:48:02 2022 +0200
|
|     adding colab notebooks
|
* commit e5c1c97fa8aaeb1f3c377a1ecb403d16140212d5
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Oct 26 11:54:47 2022 +0200
|
|     adding material lec 10
|
* commit ff60de77f3c18f8d0a8ee3b6918cd4db24bdf721
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Oct 19 11:19:43 2022 +0200
|
|     adding material lecture 9
|
* commit e6ac53804aaa4bdcac8561c891e3013a8868de3c
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Oct 5 11:32:15 2022 +0200
|
|     material lecture 7
|
* commit b412adb63d22c9f1129f195db9cdbbc63f945fe3
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Sep 28 11:49:30 2022 +0200
|
|     adding material lecture 6
|
* commit 7e0089cd9e1ab59478521d31b15867377b54008f
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Sep 28 11:06:03 2022 +0200
|
|     adding material for lecture 6
|
* commit c26010d5096e570018e4328672c45e7d54cddc69
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Sep 21 11:10:38 2022 +0200
|
|     adding material for lec 5
|
* commit 1a4a2672f3ec0b08042a76cc54654619254a0204
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Sep 14 11:34:12 2022 +0200
|
|     adding material lecture 4
|
```



```
* commit 9222545035a3bad8eec2c70774d6fad07066e113
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Tue Sep 6 14:08:22 2022 +0200
|
|     adding material for lecture 3
|
* commit 8b4efee00f04d5b3e7a4cc29269168556a9a28ad
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Tue Sep 6 11:02:01 2022 +0200
|
|     adding material for lecture 3
|
* commit 71f261f8dbb09c828dfd2be1ad664a14b1fbc498
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Wed Aug 31 09:59:14 2022 +0200
|
|     added honor code
|
* commit a610fc6f9cdf3dd3e84e42e8e6ac168a73ab3a28
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Mon Aug 29 09:33:29 2022 +0200
|
|     adding 1 asnyc exercise from lecture 1
|
* commit 0f6036be424d789df67656a3be50cec8848f2d4f
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Mon Aug 29 09:31:42 2022 +0200
|
|     adding 1 asnyc exercise from lecture 1
|
* commit ae45ae7891f40dab569153b9decaaedc71601ff5
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Mon Aug 29 08:51:21 2022 +0200
|
|     removing git_exercise.py file
|
* commit 84ed53d35856ac4c6a59a8de7853e2d723b44a39
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Fri Aug 26 08:56:46 2022 +0200
|
|     adding file for git exercise in lecture 2
|
* commit 37ac00f151ef14e398bf6ab3c3078a799d974775
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Mon Aug 22 11:57:28 2022 +0200
|
|     adding some py files for lecture 1
|
```

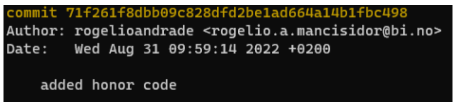


```
* commit 0fb7842d8311144c4d1941b8e9d828059e11c500
| Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
| Date:   Mon Aug 22 08:50:07 2022 +0200
|
|     adding instructions for UML
|
* commit 20b88515dc1668bed7942359e3ad183f51961f62
  Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
  Date:   Fri Aug 19 08:48:04 2022 +0200

      initial commit
(END)
```

b) We can check when a specific file was last edited with the following command line:

    git log -1 [filename]

The "-1" will make sure only the very last edit is displayed. With "README.md" as the filename we get:



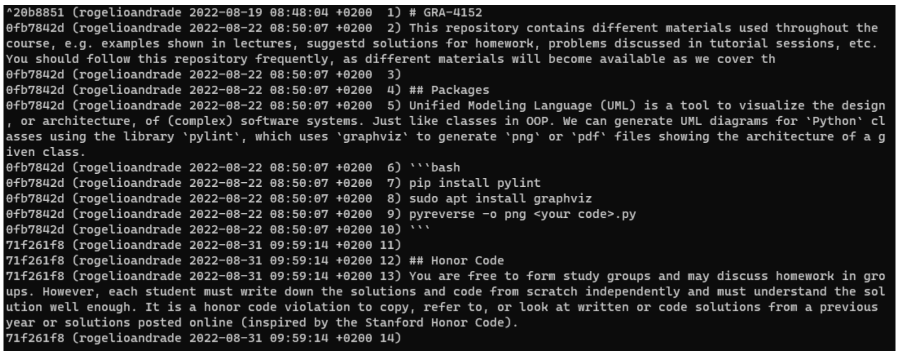
```
commit 71f261f8dbb09c828dfd2be1ad664a14b1fbc498
Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
Date:   Wed Aug 31 09:59:14 2022 +0200

    added honor code
```

Hence, README.md was last edited August 31st at 09:59:14.

c) The associated commit message associated with the last modification to the README.md is displayed when we run the command in b). As seen in the screenshot above, the commit message was "added honor code". We can confirm that this was indeed the change performed to the README.md file with the following command that shows which specific lines in the file was changed when:

    git blame README.md

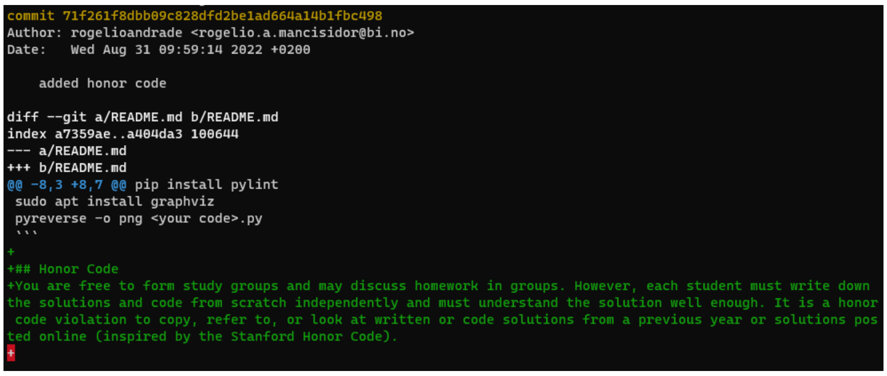
The result is:



````
^20b8851 (rogelioandrade 2022-08-19 08:48:04 +0200  1) # GRA-4152
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  2) This repository contains different materials used throughout the
course, e.g. examples shown in lectures, suggestd solutions for homework, problems discussed in tutorial sessions, etc.
You should follow this repository frequently, as different materials will become available as we cover th
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  3)
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  4) ## Packages
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  5) Unified Modeling Language (UML) is a tool to visualize the design
, or architecture, of (complex) software systems. Just like classes in OOP. We can generate UML diagrams for `Python` cl
asses using the library `pylint`, which uses `graphviz` to generate `png` or `pdf` files showing the architecture of a g
iven class.
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  6) ```bash
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  7) pip install pylint
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  8) sudo apt install graphviz
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200  9) pyreverse -o png <your code>.py
0fb7842d (rogelioandrade 2022-08-22 08:50:07 +0200 10) ```
71f261f8 (rogelioandrade 2022-08-31 09:59:14 +0200 11)
71f261f8 (rogelioandrade 2022-08-31 09:59:14 +0200 12) ## Honor Code
71f261f8 (rogelioandrade 2022-08-31 09:59:14 +0200 13) You are free to form study groups and may discuss homework in gro
ups. However, each student must write down the solutions and code from scratch independently and must understand the sol
ution well enough. It is a honor code violation to copy, refer to, or look at written or code solutions from a previous
year or solutions posted online (inspired by the Stanford Honor Code).
71f261f8 (rogelioandrade 2022-08-31 09:59:14 +0200 14)
````

We can see that indeed, the lines with the honor code were (the only) new lines in the latest commit.

To use the suggested command "git show" you need to specify the hash associated with the latest commit. It similarly shows what was added, though in a different format, here (by default) in green:



~~~
commit 71f261f8dbb09c828dfd2be1ad664a14b1fbc498
Author: rogelioandrade <rogelio.a.mancisidor@bi.no>
Date:   Wed Aug 31 09:59:14 2022 +0200

    added honor code

diff --git a/README.md b/README.md
index a7359ae..a404da3 100644
--- a/README.md
+++ b/README.md
@@ -8,3 +8,7 @@ pip install pylint
 sudo apt install graphviz
 pyreverse -o png <your code>.py
 ```
+
+## Honor Code
+You are free to form study groups and may discuss homework in groups. However, each student must write down
the solutions and code from scratch independently and must understand the solution well enough. It is a honor
 code violation to copy, refer to, or look at written or code solutions from a previous year or solutions pos
ted online (inspired by the Stanford Honor Code).
+
~~~

2.

a) Using a command-line interface like Git Bash I can add a .gitignore file with the following command:

```
touch .gitignore
```

Using the same command "touch" we can also create a foo.py file. The name of the Python file can easily be added to the .gitignore file from the command line with the following command:

```
echo "foo.py" >> .gitignore
```

We can then add the .gitignore file, commit the change and push the commit to GitHub:

```
git add .gitignore
git commit -m "added .gitignore file"
git push -u origin master
```

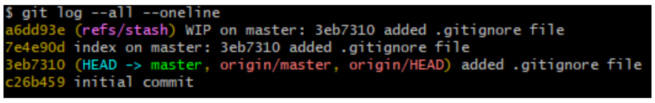
3. In this exercise I will, for simplicity, just work with my own GitHub repository. I will start by cloning the repository, then just adding a few lines to the .gitignore file:

```
git clone "https://github.com/S1042033/GRA4152"
cd "GRA4152"
```

```
echo “test.py” >> .gitignore
echo “.exe” >> .gitignore
```

a) When running the command “git stash”, git will “stash” or save the current state of the directory away and revert the working directory back to the last commit, or the “HEAD” commit. The output I got when running “git stash” is “Saved working directory and index state WIP on master: 3eb7310 added .gitignore file”.

b) I get:

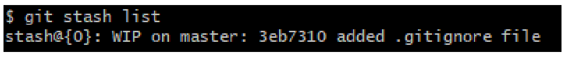


```
$ git log --all --oneline
a6dd93e (refs/stash) WIP on master: 3eb7310 added .gitignore file
7e4e90d index on master: 3eb7310 added .gitignore file
3eb7310 (HEAD -> master, origin/master, origin/HEAD) added .gitignore file
c26b459 initial commit
```

We can see that the last log entry was a stash with Work In Progress (WIP) on master, followed by the first seven characters in the hash and the commit message of the last commit.

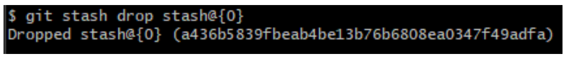
c) “git stash pop” will essentially revert what we did when running “git stash”. The stashed state will then be applied to on top of the current working directory. This can be useful if we want to keep working on what we “stashed away”, e.g., if we used stash as a temporary save.

d) If we run “git stash” again and then run “git stash list” we can list all our current stashes:



```
$ git stash list
stash@{0}: WIP on master: 3eb7310 added .gitignore file
```

Here we see we only have one stash, with stash id “stash@{0}”. We can then run:



```
$ git stash drop stash@{0}
Dropped stash@{0} (a436b5839fbeab4be13b76b6808ea0347f49adfa)
```

“git stash list” will now return nothing.

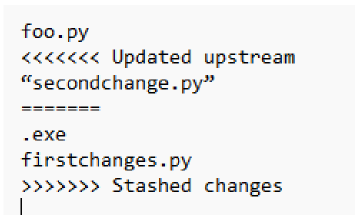
Let’s start by making some changes to the .gitignore file and stash the changes:

```
echo “.exe” >> .gitignore
echo “firstchanges.py” >> .gitignore
git stash
```

Then let’s make a different change and commit the changes:

```
echo “secondchange.py” >> .gitignore
git commit -m "changes to .gitignore"
```

When running “git stash pop” I get the warning “CONFLICT (content): Merge conflict in .gitignore”. In the .gitingore file I can see that git has merged the two changes:



```
foo.py
<<<<<<< Updated upstream
“secondchange.py”
=======
.exe
firstchanges.py
>>>>>>> Stashed changes
```

4. We can add a branch with:

```
git checkout -b my_test_branch
```

a) We can move between them with “git checkout [name of branch]”, here name would be master of my_test_branch

b) Let’s add a line to README.md and commit:

```
echo "test_line_new_branch" >> README.md
git add README.md
git commit -m "changes in new branch"
```

c) I then merge the branch:

```
git merge my_test_branch
```