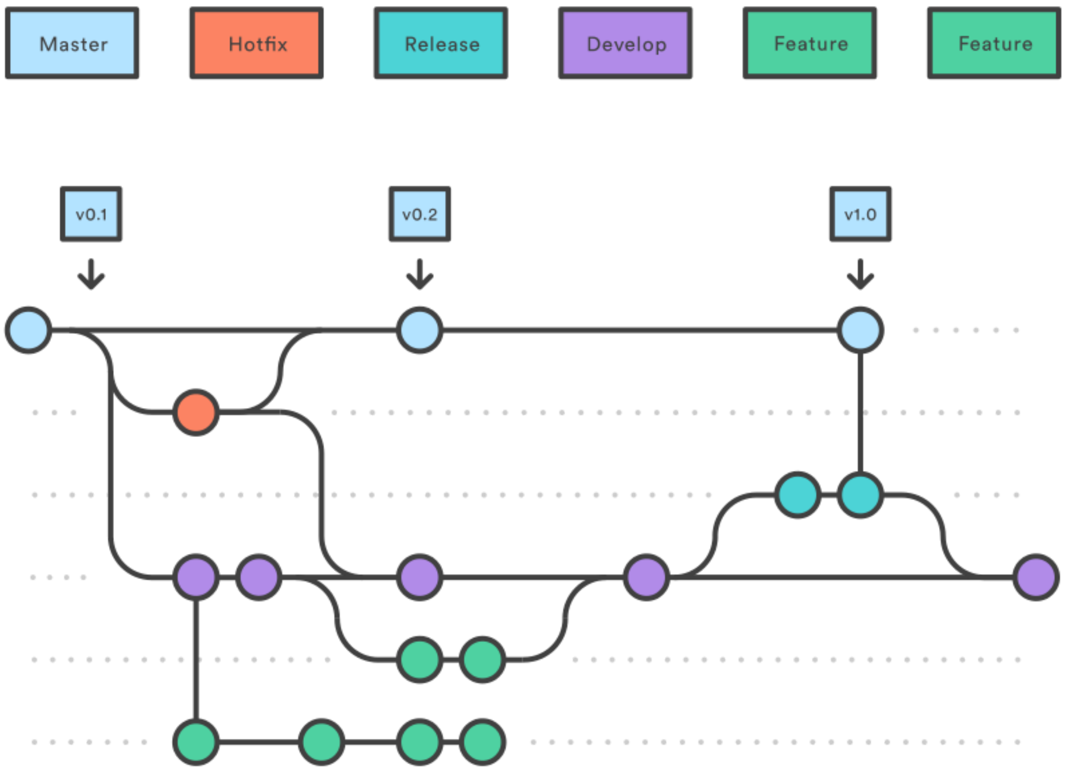
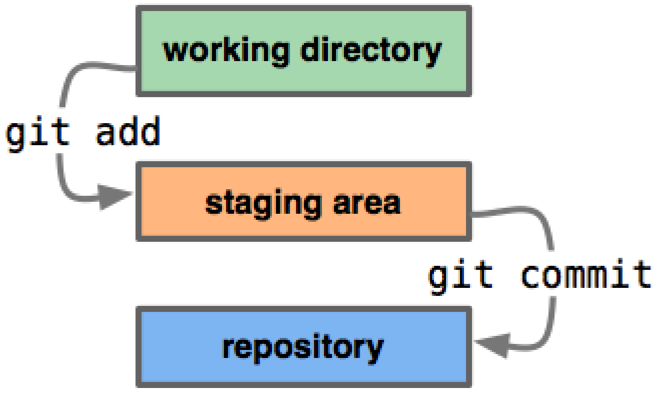
UD03: Git Exercises



1. Areas of a Git repository

In this activity you will have to search for information and explain the 3 areas of a Git project:

- **Working directory**
- **Staging area**
- **Repository** (**.git folder)**



Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

2. Inicialize local repository

In the next activity, we are going to create a local repository, that is, on our personal PC. Then we will add and modify some files and log the changes. We will work from the text terminal.

You must collect the information and captures necessary to generate the final document.

We will follow the following process:

2.1. We create a folder to host the proyect.

For example we can do (put yor name).

```
mkdir pruebas-david
cd pruebas-david
```

2.2. We check that we have the empty folder

```
ls -la
```

2.3. We initialize the repository

```
git init
```

2.4. We check that a folder has been created `.git`.

This is the folder where all the changes we make will be recorded.

```
ls  -la
```

Check the content of this new folder, why does it have a dot in front of it?

2.5. Create/edit a file `README.md`

```
nano README.md
```

2.6. We add a line to the file

We add to this file a line with our name and surnames. We save file.

2.7. We register changes in the repository.

To do this we must perform 2 steps:

1. Added to staging area:

```
git add README.md
```

2. Add to repository:

```
git commit -m "Primer cambio registrado David"
```

The first command (`git add`) ads the `README.md` file to the staging area. And the second command (`git commit ...`) adds it to the local repository.

2.8. We repeat points 5, 6 and 7 two more times.

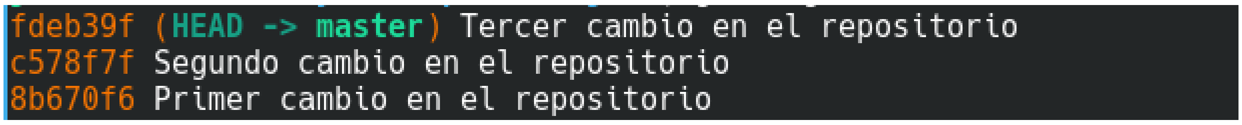
The first time we add a line with the current date and then we do again `git add ...` y `git commit ...`

The second time we add a line with the name of the IES and then we do again `git add ...` y `git commit ...`

2.9. Finally we see changes made

For them we will execute

```
git log
git log --oneline
```



```
fdeb39f (HEAD -> master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 Primer cambio en el repositorio
```

3 commits should appear

NOTE: Do not delete the local repository. We will use it again in the next activity.

Upload a PDF document to the AULES platform with the relevant screenshots and explanations.

3. Review commits made

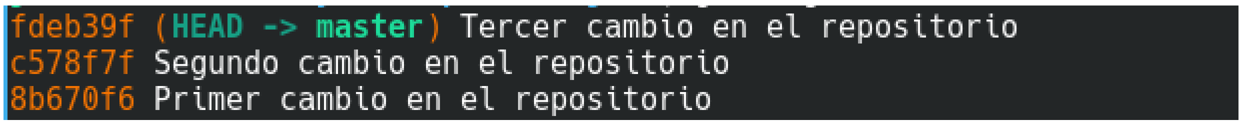
In the next activity, we will use the command `git checkout` to move through the different commits First of all check that you have at least 3 commits made. To do this, run:

```
git  log  --oneline --all
```

The option `--oneline`, It shows us the information of each commit in one line.

The option `--all`, shows us all the commits.

You should see something similar to the following image:



```
fdeb39f (HEAD -> master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 Primer cambio en el repositorio
```

The first column is a **hash**,an identifier.

The numbers are not ordered. In my case the first commit has a hash `8b670f6`. My last commit is `fedb39f`.

You will have a different hash. Don't worry, it is.

The second column is the message that we put when we made the commit.

Also notice that in the last commit, in my case `fedb39f`, there is **identifier HEAD**.
This is a reference that points to the commit we are currently on.
In addition, another **mastes identifier** appears that indicates the branch we are on. By default, it is always master

The master identifier always points to the last commit of the branch.
However, we can move the HEAD identifier and move between different commits and see how the files were at each moment.

To move the HEAD identifier we use the command **git checkout** *hash_number*.

Perform the following steps and create the corresponding captures:

3.1. Let's see the content of the `README.md` file in the current commit.

For this we do:

```
cat  README.md
```

3 lines of text should appear: your name, the date and the IES.

3.2. Let's move to the first commit..

For this we do:

```
git checkout 8b670f6
```

You must put the hash you have in the first commit.

You will see a message containing "*You are in 'detached HEAD' state...*". This indicates that the HEAD reference is not at the end of the branch. Do not worry about it.

Now let's see the content of the file `README.md`.

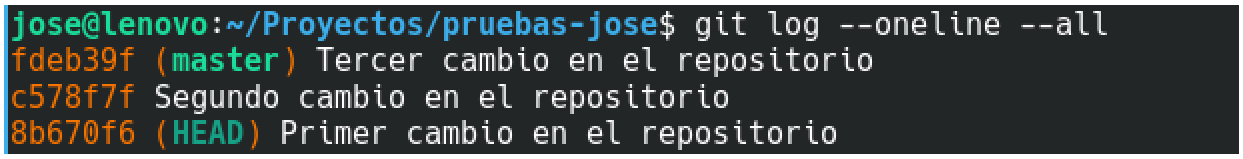
```
cat README.md
```

Only one line should appear with your name. It is the content that said file had in that commit.

3.3. Let's see where we are on the branch. For this we execute:

```
git log --oneline --all
```

Something similar to the following image should appear:



```
jose@lenovo:~/Proyectos/pruebas-jose$ git log --oneline --all
fdeb39f (master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (HEAD) Primer cambio en el repositorio
```

Notice where the HEAD reference is pointing at this time.

Something that may have gone unnoticed but is extremely IMPORTANT is that every time we move from one commit to another, the content of the working directory changes. This is done automatically by git.

We WILL NOT make any changes to the files, we are just going to take a look.

3.4. Let's move to the second commit.

For this we do:

```
git checkout c578
```

You must put the hash you have. It is not necessary to put all the digits, we can shorten the hash.

Run

```
cat README.md
```

and take a screenshot.

There should be 2 lines: your name and the date.

3.5. Do it again

```
git  log  --oneline --all
```

And check that `HEAD` it's in the second commit.

3.6. Finally,

To go back to the last commit of the master branch, we simply do:

```
git checkout master
```

We can see that everything is in its place by

```
git log  --oneline --all
```

Take a screenshot.

NOTE: Do not delete the local repository. We will use it again in the next activity.

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

4. Tag commits and see differences

In this activity we are going to see 3 commands:

- `git tag`
- `git show`
- `git diff`

The first command (`git tag`) allows us to put labels on commits. **Not all commits are tagged, only the releases we want**.

The next 2 (`git show` y `git diff`) are to see the changes made between different commits. They are very similar although with small differences.

Basically `git show` allows us to see the changes of a commit with respect to the previous one, while `git diff` allows us to see changes in a range of commits.

However, both `git show` and `git diff` have so many options that we will only focus on the essential ones here.

Let us begin.

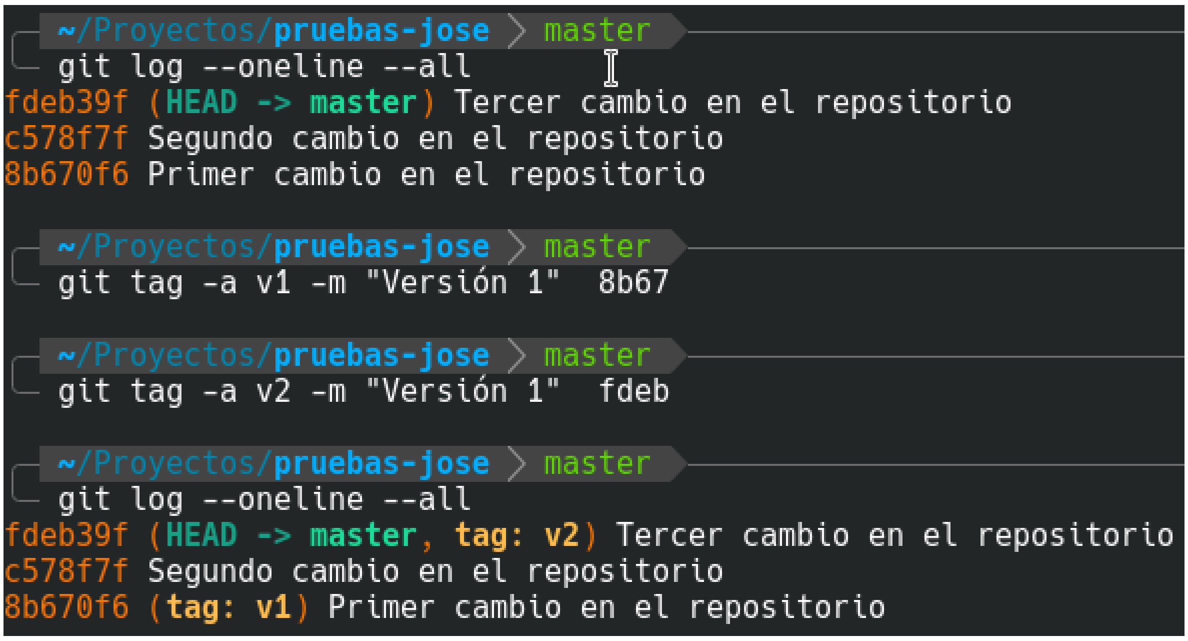
4.1. We label the commit first and the third.

The first commit will be version 1 of our project. The tag will be `v1`.

The third commit will be version 2 of our project. The tag will be `v2`.

The second commit will not be tagged.

The screenshot shows an error that we will later correct in the v2 tag.



```
~/Proyectos/pruebas-jose > master
git log --oneline --all
fdeb39f (HEAD -> master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 Primer cambio en el repositorio

~/Proyectos/pruebas-jose > master
git tag -a v1 -m "Versión 1"  8b67

~/Proyectos/pruebas-jose > master
git tag -a v2 -m "Versión 1"  fdeb

~/Proyectos/pruebas-jose > master
git log --oneline --all
fdeb39f (HEAD -> master, tag: v2) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

To tag we use the command

```
git tag -a nombre_etiqueta -m "Mensaje"  commit_a_etiquetar
```

For example, in my case:

```
git tag  -a v1  -m "Versión 1"  8b67
git tag  -a v2  -m "Versión 2"  fdeb
```

The `-a` option means annotate.

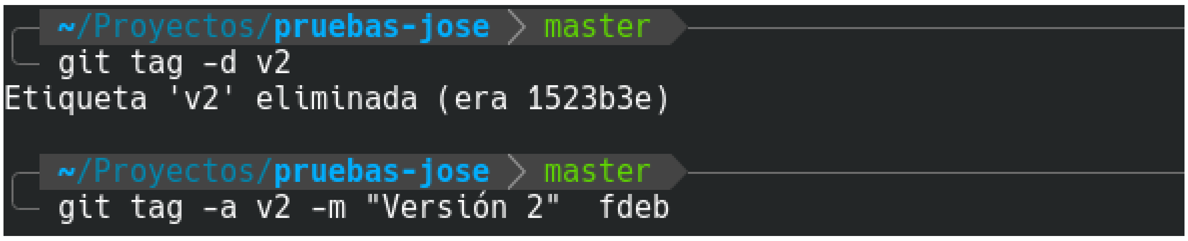
The `-m` option allows us to put a message.

Finally we must put the commit to which we want to apply the label.

If for any reason we make a mistake when creating the label, we can eliminate it with

```
git tag -d nombre_etiqueta
```

For example, I made a mistake in the message of v2, so I do



```
~/Proyectos/pruebas-jose > master
 git tag -d v2
Etiqueta 'v2' eliminada (era 1523b3e)

~/Proyectos/pruebas-jose > master
 git tag -a v2 -m "Versión 2"  fdeb
```

4.2. Using labels to move

Tags allow us to reference commits in a more convenient way than using the hash identifier.

For example it is more comfortable to use:

```
git checkout v1
```

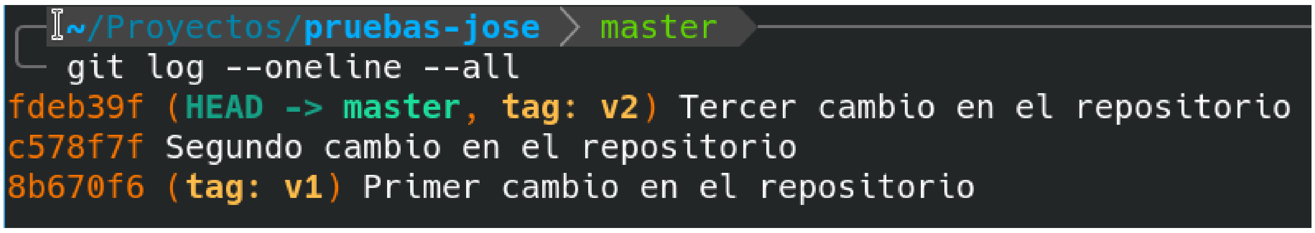
what to wear

```
git checkout 8b67
```

To go back to the last commit do

```
git checkout master
```

4.3. Examined changes of a commit against the previous one.



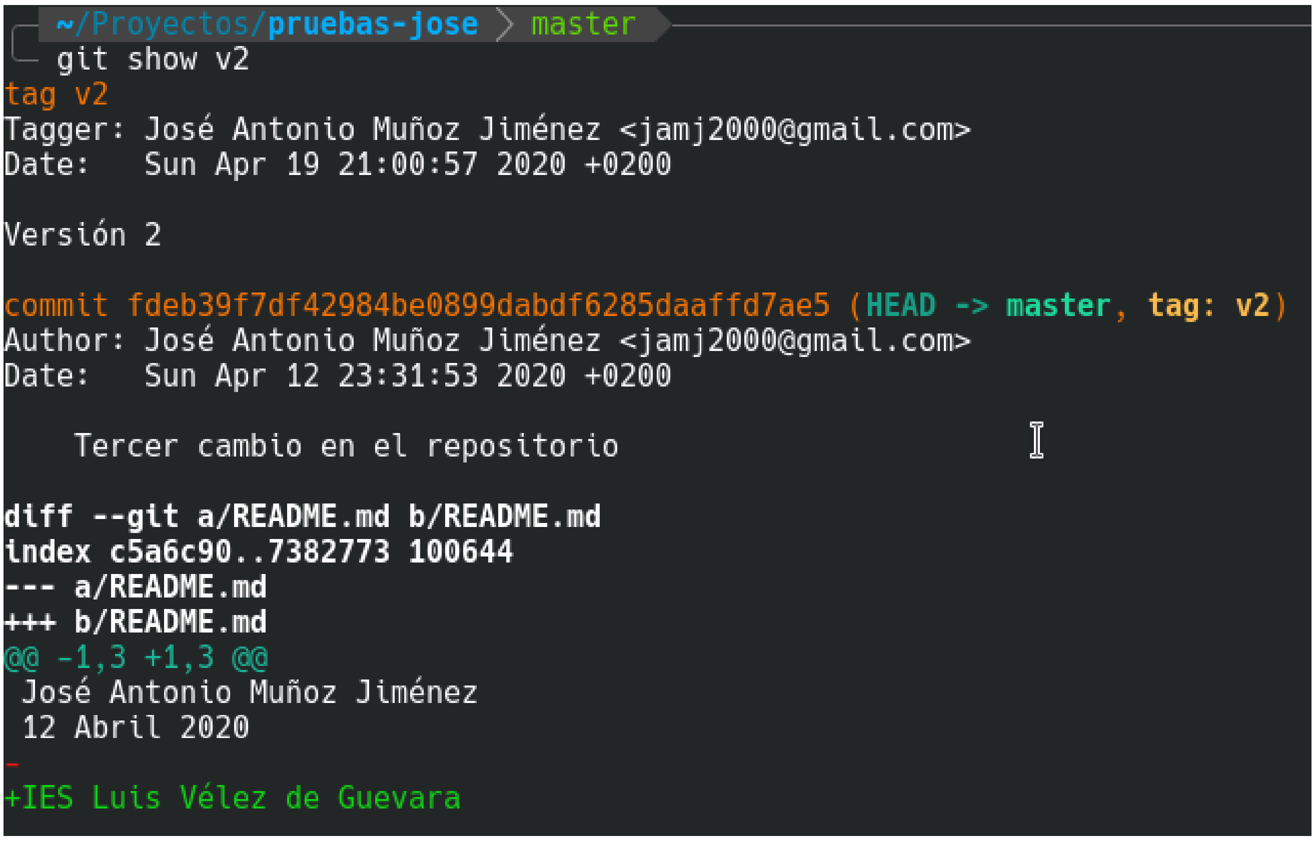
```
~/Proyectos/pruebas-jose > master
 git log --oneline --all
fdeb39f (HEAD -> master, tag: v2) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

To see the changes introduced with respect to the previous commit we do:

```
git show
```

In this case, by matching all pointers (HEAD, master, v2, and fdeb) to the same site, the above command is equivalent to

```
git show HEAD
git show master
git show fdeb
git show v2
```



```
~/Proyectos/pruebas-jose > master
git show v2
tag v2
Tagger: José Antonio Muñoz Jiménez <jamj2000@gmail.com>
Date:   Sun Apr 19 21:00:57 2020 +0200

Versión 2

commit fdeb39f7df42984be0899dabdf6285daaffd7ae5 (HEAD -> master, tag: v2)
Author: José Antonio Muñoz Jiménez <jamj2000@gmail.com>
Date:   Sun Apr 12 23:31:53 2020 +0200

    Tercer cambio en el repositorio

diff --git a/README.md b/README.md
index c5a6c90..7382773 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,3 @@
 José Antonio Muñoz Jiménez
 12 Abril 2020
-
+IES Luis Vélez de Guevara
```

As we can see, a line was added, the one that contains the IES.

Added lines appear in green with a `+` sign.

Deleted lines appear in red and with a `-` sign.

In this case we have only performed addition operations.

To see the change made in the second commit compared to the first, we do

```
git show c578
```

The line with the date should be added.

And to see the change made in the commit first relative to the empty repository, we do

```
git show v1
```

The line with the name should appear added.

4.4. Examined changes of a commit with respect to several previous ones.

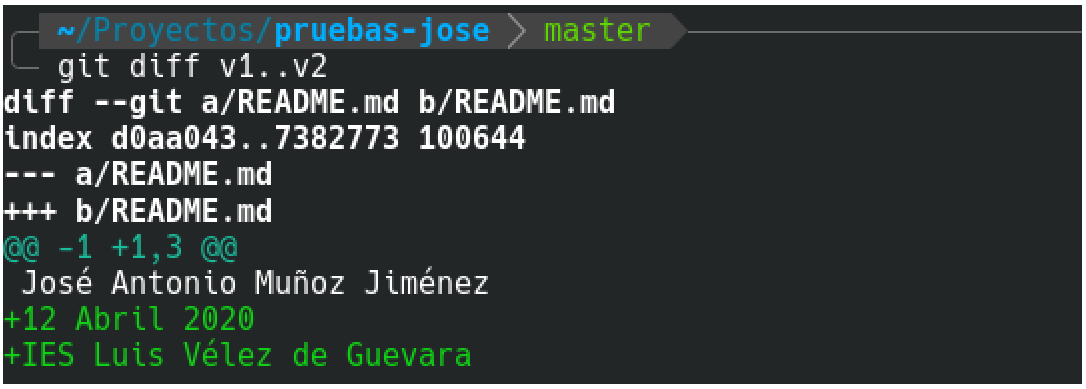
If we want to see all the changes made over several commits, we will use `git diff`.

The way of use is

```
git diff commit1..commit2
```

For example, to see the changes between version 1 and version 2, we do

```
git diff v1..v2
```



```
~/Proyectos/pruebas-jose 〉 master
  git diff v1..v2
diff --git a/README.md b/README.md
index d0aa043..7382773 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,3 @@
 José Antonio Muñoz Jiménez
+12 Abril 2020
+IES Luis Vélez de Guevara
```

We can see that 2 lines have been added since commit v1.

It is highly recommended to put the oldest commit first and then the newest commit. If we do it the other way around, instead of appearing in green, the result will appear in red, and its interpretation will be more confusing.

4.5. Difference between `git show` and `git diff`

we can also do

```
git show v1..v2
```

Run that command and take a screenshot. Briefly explain the difference between `git diff v1..v2`

NOTE: Do not delete the local repository. We will use it again in the next activity.

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

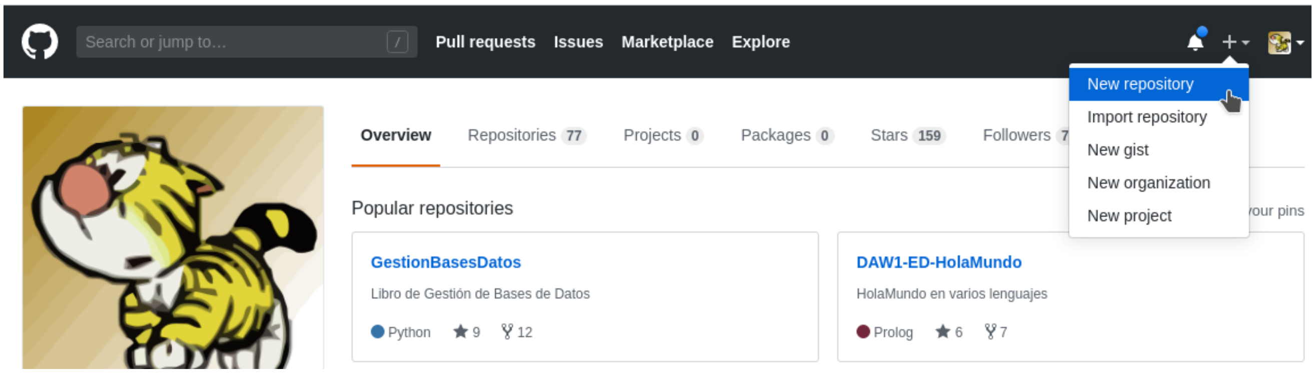
5. Create remote repository and upload local commits

In this activity we will create an empty repository on GitHub and upload the content of our local repository.

5.1. First, we create a completely empty repository on GitHub.

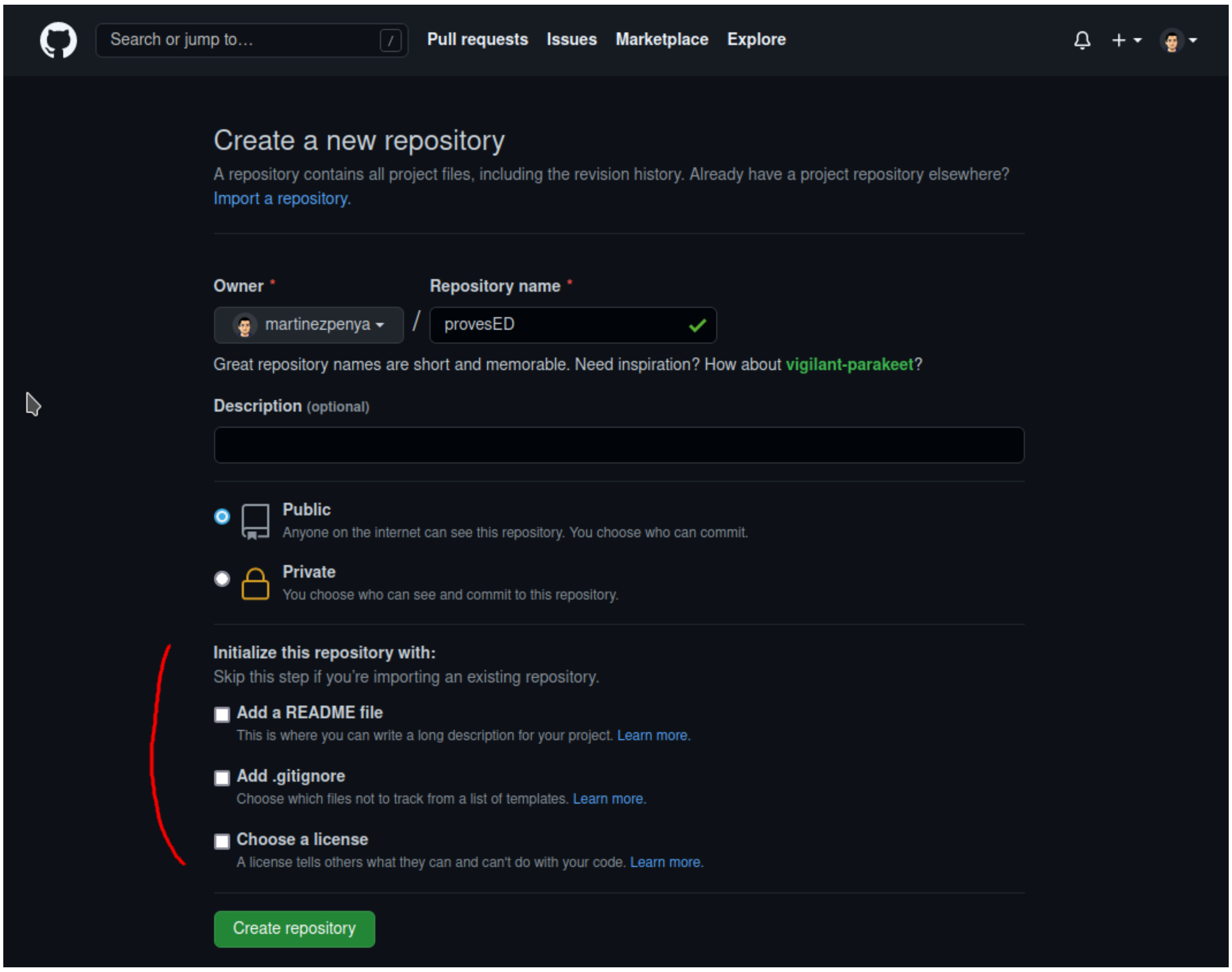
We access our GitHub account.

In the **upper right corner**, click on the **+** sign and then on **New repository**



We choose the name of the repository. It does not have to match the name of the local repository, although it is advisable so as not to make a mess.

Instead of provesED put your name.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * martinezpenya / Repository name * provesED

Great repository names are short and memorable. Need inspiration? How about vigilant-parakeet?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

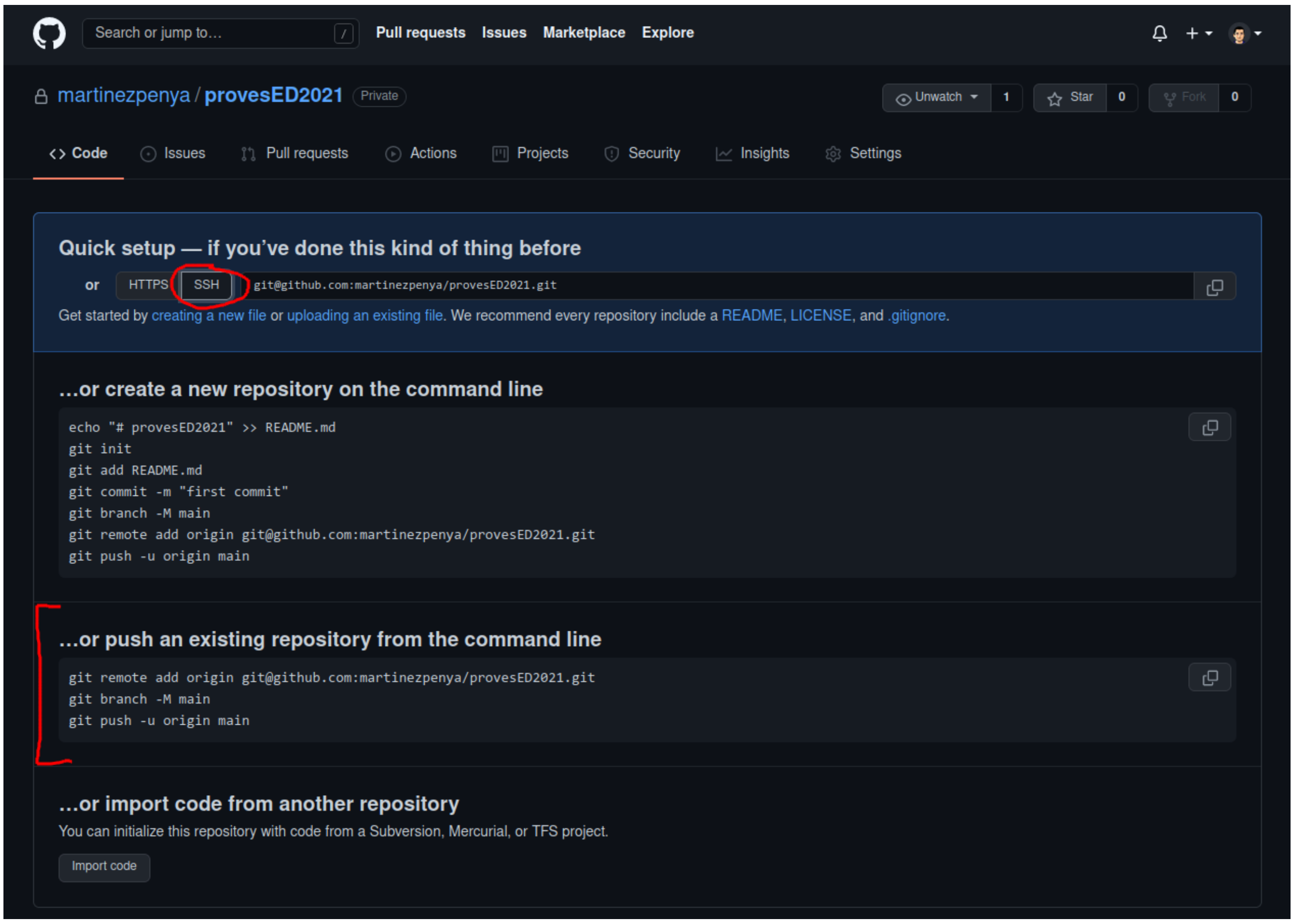
- [ ] Add a README file — This is where you can write a long description for your project. Learn more.
- [ ] Add .gitignore — Choose which files not to track from a list of templates. Learn more.
- [ ] Choose a license — A license tells others what they can and can't do with your code. Learn more.

You can choose to your liking if the repository is public or private, this will not affect the rest of the sections.

It is very important that you **DO NOT INITIALIZE THE REPOSITORY**. If the repository was not empty it could give us a conflict.

In a later activity we will create conflicts and see how to resolve them. But in this activity, we are only going to work on the basics.

We will click on **Create Repository** and a page like the following will appear:



Quick setup — if you've done this kind of thing before

or HTTPS SSH `git@github.com:martinezpenya/provesED2021.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# provesED2021" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:martinezpenya/provesED2021.git
git push -u origin main
```

…or push an existing repository from the command line

```
git remote add origin git@github.com:martinezpenya/provesED2021.git
git branch -M main
git push -u origin main
```

…or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

There we can see the URL of the remote repository. There are 2 ways to access:

- **via HTTPS**
- **via SSH**

We will use SSH since it is more secure and allows us to use public-private encryption due to the fact that recently github has disabled access by username and password. In the point `2.1 Configuration with public/private key` of the file `UD03_anexo_ES.pdf` you have detailed the configuration and steps to follow, if you have not yet configured your PC in this way... **you must do it before continuing** .

Below are the commands to execute in our local repository. We see it in the next point.

For your comfort, do not close the page. We will come back to it later.

5.2. Associate local repository with remote repository

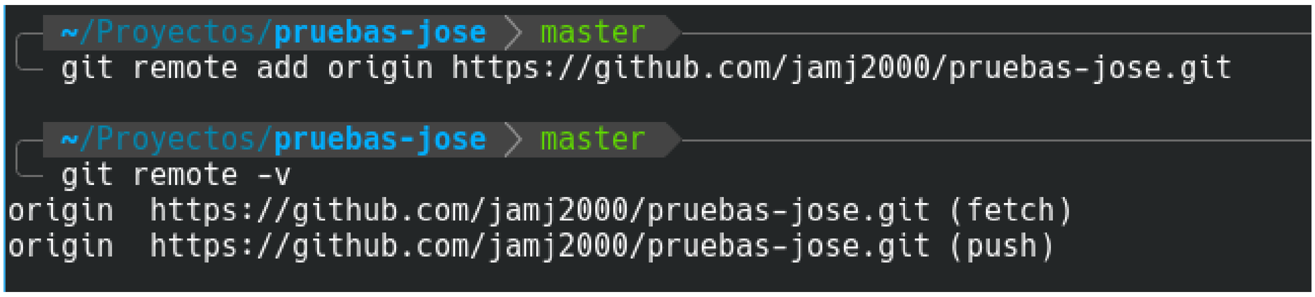
In our local repository, to associate it with the remote repository, we do:

```
git remote add origin git@github.com:martinezpenya/provesED2021.git
```

Our remote repository will be identified as **origin**. We can give it another name, but we must not. It is a widely accepted convention to name the remote GitHub repository by this name.

To see if it has been added correctly we do

```
git remote -v
```



```
~/Proyectos/pruebas-jose > master
git remote add origin https://github.com/jamj2000/pruebas-jose.git

~/Proyectos/pruebas-jose > master
git remote -v
origin  https://github.com/jamj2000/pruebas-jose.git (fetch)
origin  https://github.com/jamj2000/pruebas-jose.git (push)
```

2 entries should appear, one for download (fetch) and one for upload (push)

NOTE: If for any reason we make a mistake and misspell the name or URL, we can delete the association with

```
git remote remove origin
```

and then recreate the association.

5.3. Push all local commits to remote repository

To upload the content of our local repository to the remote repository we do:

```
git push -u origin master
```

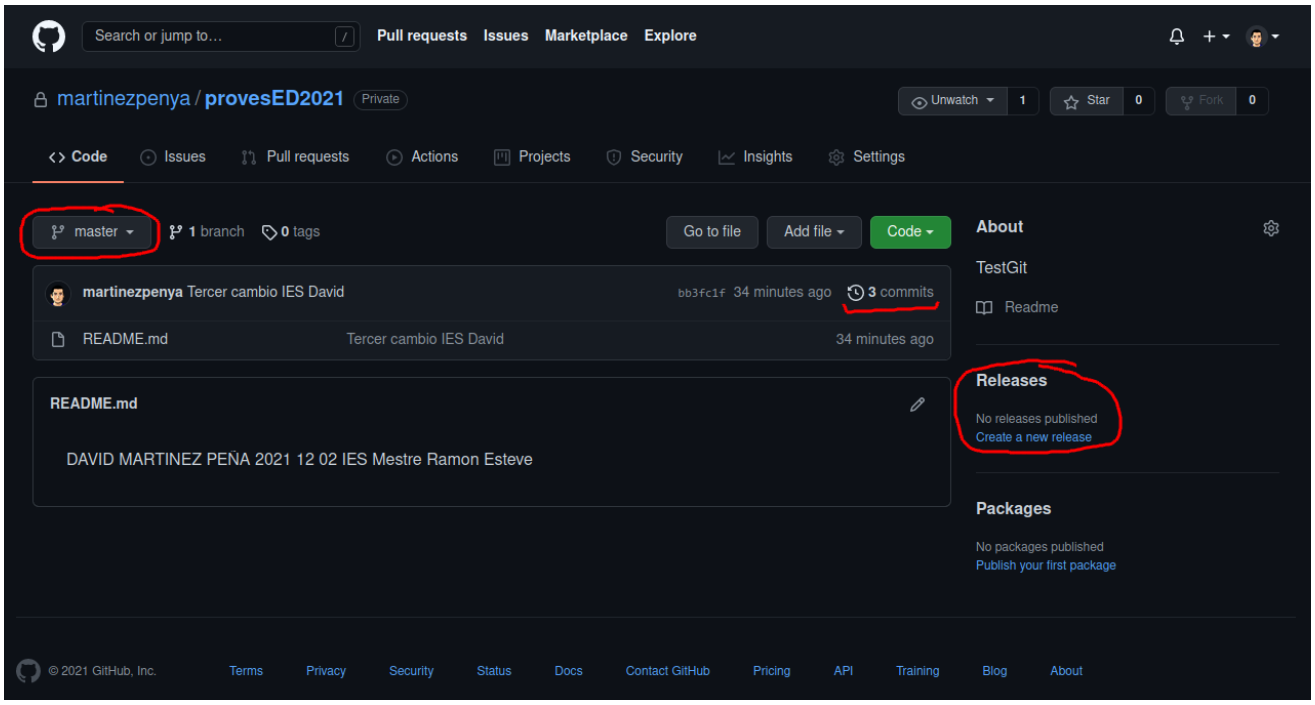
The **origin** identifier is the name we gave our link. The identifier **master** refers to the main branch.

It's a widely followed convention, so stick to it.

If we have correctly configured git on our PC, the changes from our PC should be sent to the remote repository without asking for a password since we are using the key that we have configured in our system.

5.4. Checking the upload.

We go back to the GitHub page and update it. Something similar to this will appear:



martinezpenya Tercer cambio IES David bb3fc1f 34 minutes ago 3 commits

README.md Tercer cambio IES David 34 minutes ago

README.md

DAVID MARTINEZ PEÑA 2021 12 02 IES Mestre Ramon Esteve

GitHub offers many features.

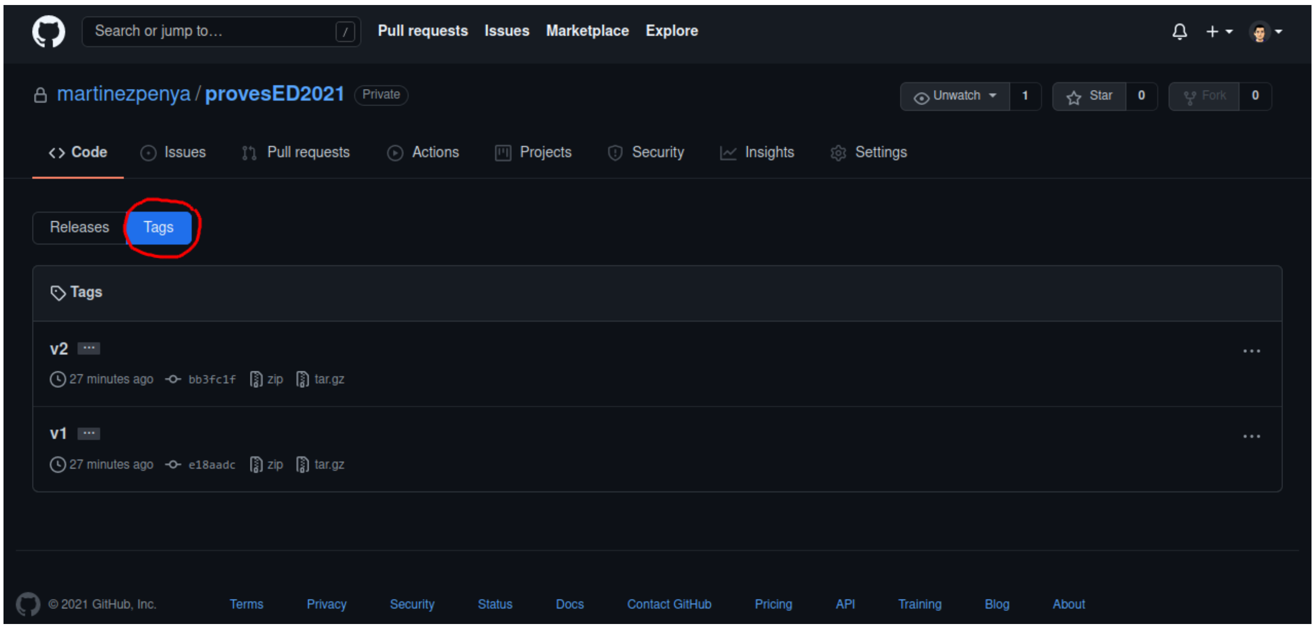
So we'll focus on releases right now. These correspond to the tagging we did in the previous activity with `git tag`.

We had 2 releases, labeled v1 and v2, but none appear here.

The reason is that we must upload the labels separately with the command

```
git push --tags
```

So we will execute said command from our local repository. We will refresh the page. Et voilà!



v2
27 minutes ago bb3fc1f zip tar.gz

v1
27 minutes ago e18aadc zip tar.gz

5.5. Examining commits and releases on GitHub.

Tap on commits and take a screenshot. On your own you can examine each of the commits.

Tap on Tags and take a screenshot. Note that compressed files have been created with the source code for download.

NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.

Upload a PDF document to the AULES platform with the relevant screenshots and explanations.

6. Undo changes in local repository

In this activity, we will see what we can do when we make mistakes.

If we make a change and we have "messed up", we can undo the "wrong".

Let's see it in a practical way using the `git reset --hard` command.

6.1. Undo changes to the working directory

Being in the last commit of the master branch, we modify the `README.md` file

Let's remove the last 2 lines.

```
nano README.md
```

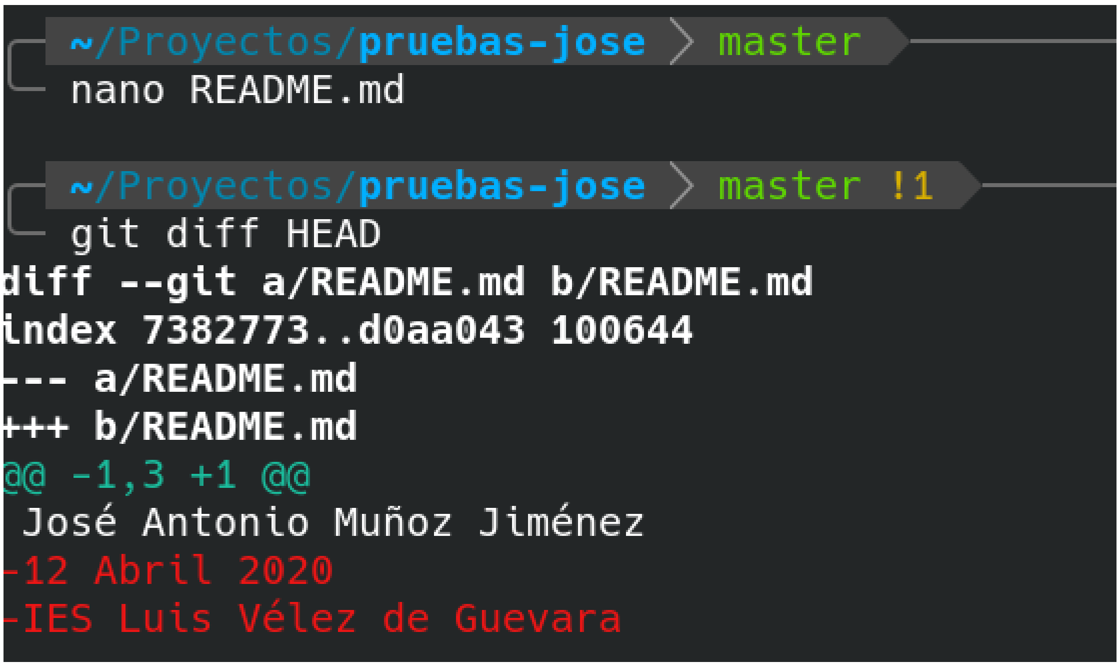
We edit. There should be a single line with our name.

To see the changes that we have introduced, we execute:

```
git diff HEAD
```

In other words, we are going to see the differences that exist in our working directory with respect to the HEAD commit, that is, the last confirmed commit.

NOTE: If we wanted to see the differences in our working directory from the Version 1 commit, we would do `git diff v1`.
Notice that we are seeing the differences towards the past. This way of using git diff is different from the one we saw in the last activity, in which we saw the differences going forward.

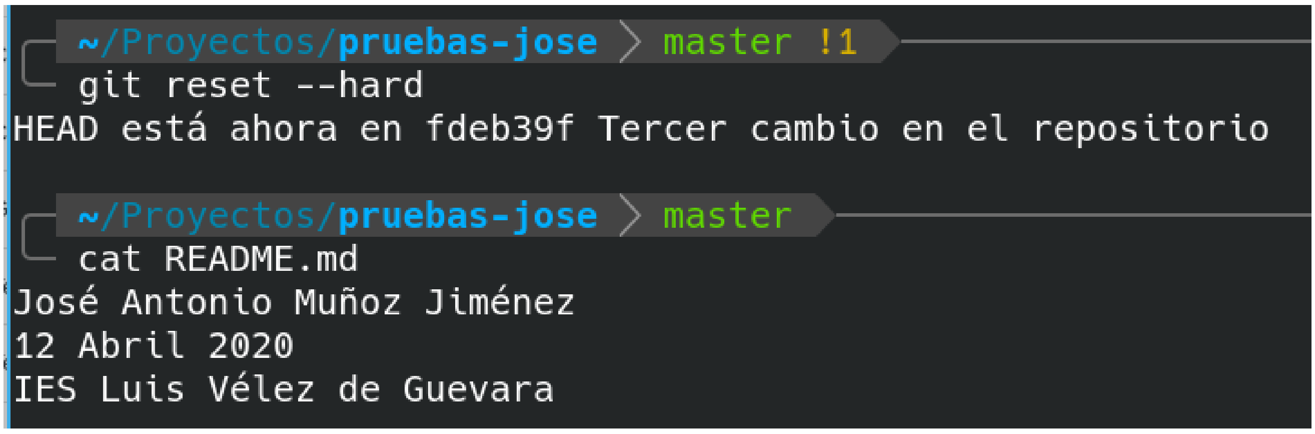


```
~/Proyectos/pruebas-jose  master
nano README.md

~/Proyectos/pruebas-jose  master !1
git diff HEAD
diff --git a/README.md b/README.md
index 7382773..d0aa043 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1 @@
 José Antonio Muñoz Jiménez
-12 Abril 2020
-IES Luis Vélez de Guevara
```

It is clearly seen that we have removed the last 2 lines.

To return the status of this file and ANY OTHER in our working directory that we have modified, we run:

```
git reset --hard
```



```
~/Proyectos/pruebas-jose  master !1
git reset --hard
HEAD está ahora en fdeb39f Tercer cambio en el repositorio

~/Proyectos/pruebas-jose  master
cat README.md
José Antonio Muñoz Jiménez
12 Abril 2020
IES Luis Vélez de Guevara
```

6.2. ¿And to undo the preparation area?

Let's imagine that we have gone a little further, and that in addition to modifying the working directory, we have added the changes to the Staging Area. That is, we have done

```
nano README.md
```

Deleted the last 2 lines.

And then we added to the staging area by

```
git add README.md
```

Do not worry in this case the previous command can also be applied:

```
git reset --hard
```

This command takes the contents of our committed commit and retrieves both the working directory and the staging area.

6.3. And what if I already made a commit?

Let's imagine that we have gone a little further, and that in addition to modifying the working directory and adding the changes to the Staging Area, we have made a commit. That is, we have done

```
nano README.md
```

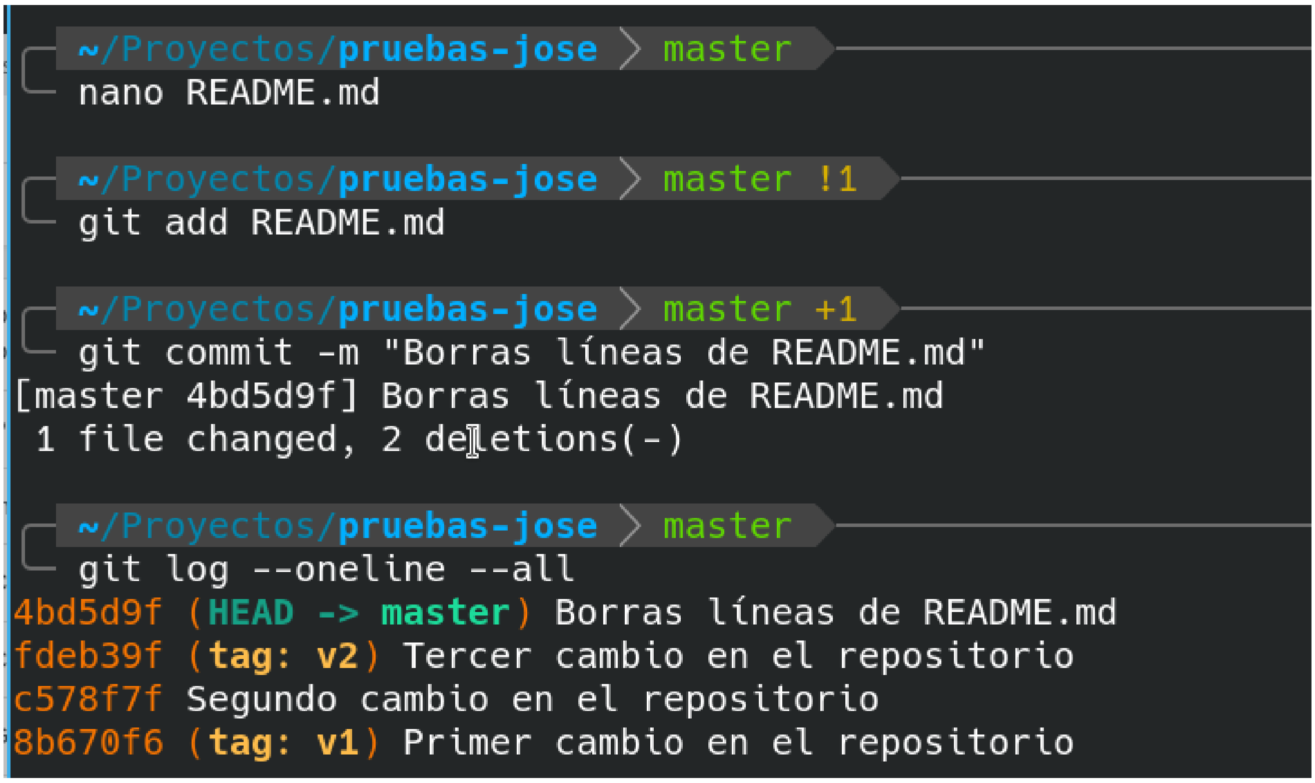
Deleted the last 2 lines.

And then we added to the staging area by

```
git add README.md
```

And we have also done

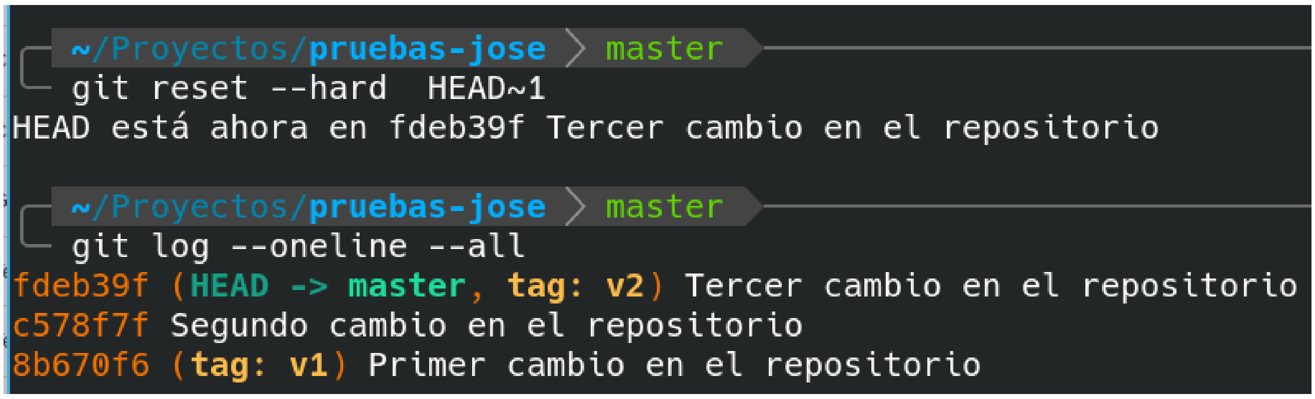
```
git commit -m "Delete lines from README.md"
```



```
~/Proyectos/pruebas-jose > master
nano README.md

~/Proyectos/pruebas-jose > master !1
git add README.md

~/Proyectos/pruebas-jose > master +1
git commit -m "Borras líneas de README.md"
[master 4bd5d9f] Borras líneas de README.md
 1 file changed, 2 deletions(-)

~/Proyectos/pruebas-jose > master
git log --oneline --all
4bd5d9f (HEAD -> master) Borras líneas de README.md
fdeb39f (tag: v2) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

Well, in this case we can also use the `git reset --hard` command as follows:

```
git reset --hard HEAD~1
```



```
~/Proyectos/pruebas-jose > master
git reset --hard  HEAD~1
HEAD está ahora en fdeb39f Tercer cambio en el repositorio

~/Proyectos/pruebas-jose > master
git log --oneline --all
fdeb39f (HEAD -> master, tag: v2) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

HEAD~1 means the commit before the current one. That is **one commit backwards**.

HEAD~2 means **2 commits backwards**.

HEAD~n means **n commits backwards**, substituting a number for n.

NOTE: Using `git reset --hard` in the latter way is dangerous, because we lose the last commit(s). So we have to make sure that this is what we want.

NOTE: Do not delete the local or remote repository. We will use them again in the next activity.

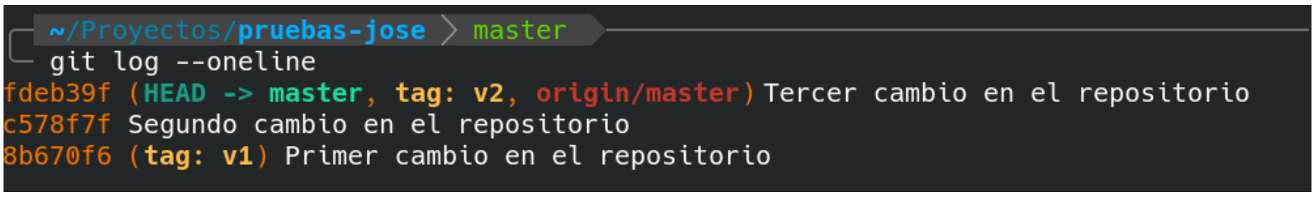
Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

7. `.gitignore` file

In this activity we will start working with something more real. For example, a simple Java application. This activity is also practical.

We are going to continue using the repository that we were using in the previous activities.

```
git log  --oneline --all
```

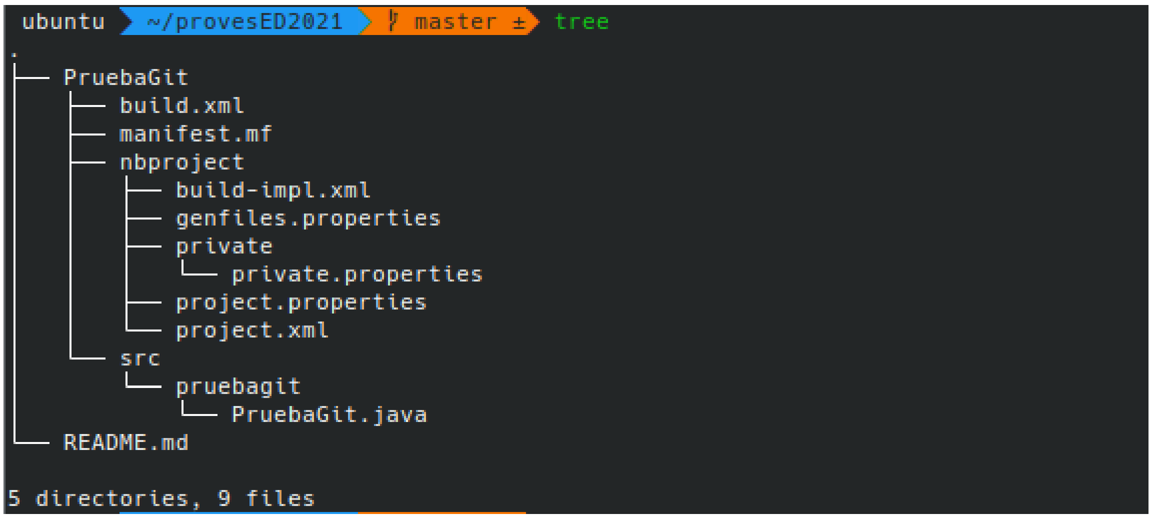


```
~/Proyectos/pruebas-jose  master
git log --oneline
fdeb39f (HEAD -> master, tag: v2, origin/master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

7.1. We create a HelloWorld application in Java with our IDE.

To do this we will open our favorite IDE (in my case NetBeans) we will create a new project (in my case TestsGit) based on ant in the same folder where we have our local GIT repository. We create the main class, and modify it so that it can print the typical "Hello world."

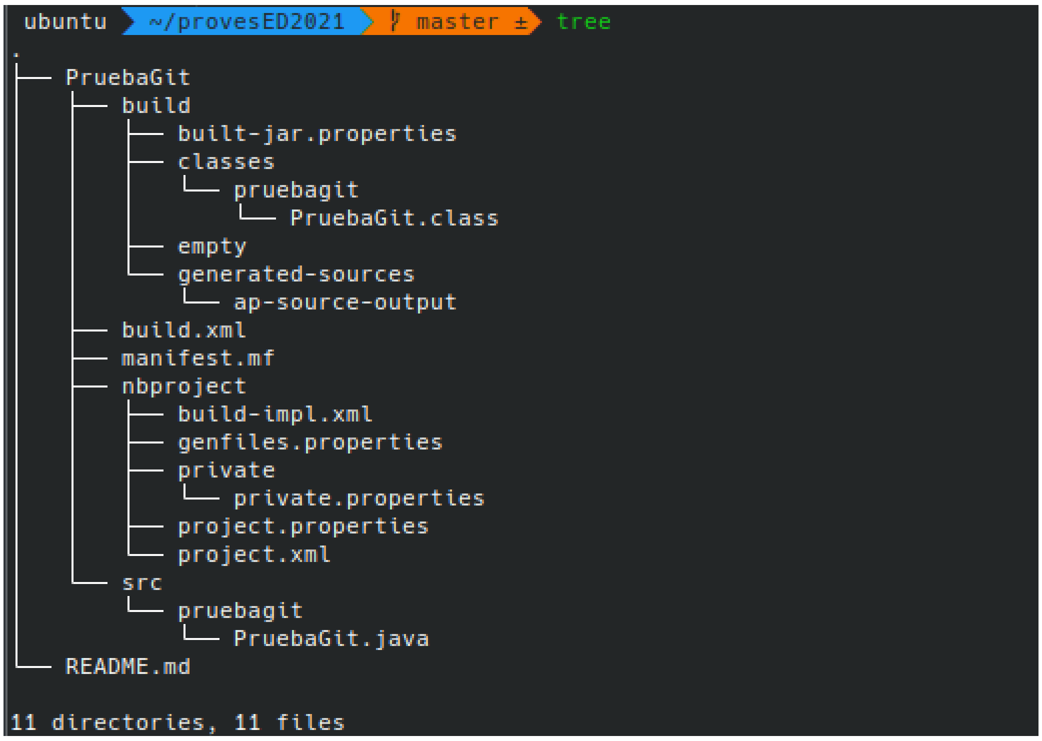
Our folder structure should look something like this:



```
ubuntu  ~/provesED2021  master ±  tree
.
├── PruebaGit
│   ├── build.xml
│   ├── manifest.mf
│   ├── nbproject
│   │   ├── build-impl.xml
│   │   ├── genfiles.properties
│   │   ├── private
│   │   │   └── private.properties
│   │   ├── project.properties
│   │   └── project.xml
│   └── src
│       └── pruebagit
│           └── PruebaGit.java
└── README.md

5 directories, 9 files
```

NOTE: The `README.md` file is not created by NetBeans. It already existed previously in the folder.

If now from NetBeans we compile or run our program, a new folder will also appear called `build`:



```
 ubuntu  ~/provesED2021  master ±  tree
.
├── PruebaGit
│   ├── build
│   │   ├── built-jar.properties
│   │   ├── classes
│   │   │   └── pruebagit
│   │   │       └── PruebaGit.class
│   │   ├── empty
│   │   └── generated-sources
│   │       └── ap-source-output
│   ├── build.xml
│   ├── manifest.mf
│   ├── nbproject
│   │   ├── build-impl.xml
│   │   ├── genfiles.properties
│   │   ├── private
│   │   │   └── private.properties
│   │   ├── project.properties
│   │   └── project.xml
│   └── src
│       └── pruebagit
│           └── PruebaGit.java
└── README.md

11 directories, 11 files
```

We can see in the list that inside the `build` folder we have the bytecode resulting from the compilation.

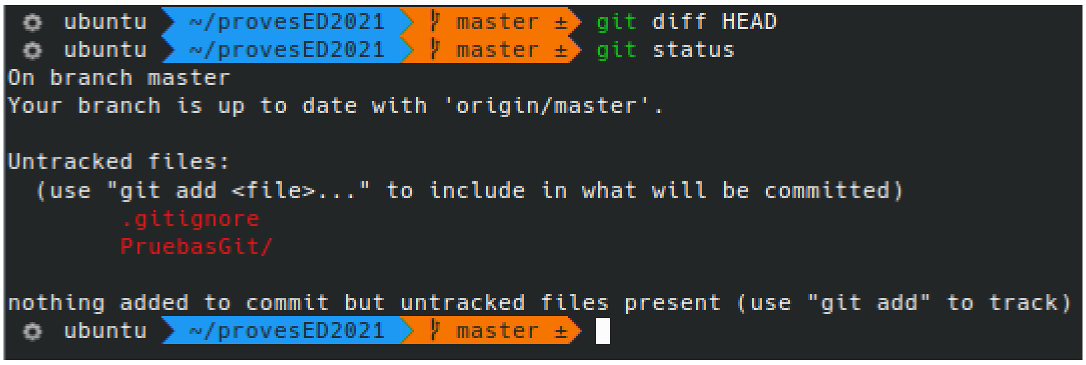
7.2. Adding files to the local repository

As we saw in the previous activity, if we now run `git diff HEAD`, we would expect to see our working directory change from the last commit.

However this is not what happens. **NOTHING DISPLAYED**. Why is this?

This is because git diff `HEAD` works always taking into account files that have already been previously added to the repository. In other words, it only takes into account the files with monitoring.

New files are untracked files. In this case we must use `git status` to see this circumstance.



```
 ubuntu  ~/provesED2021  master ±  git diff HEAD
 ubuntu  ~/provesED2021  master ±  git status
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)
        .gitignore
        PruebasGit/

nothing added to commit but untracked files present (use "git add" to track)
 ubuntu  ~/provesED2021  master ± 
```

Now we need to add all these files to the staging area and then commit.

BUT WAIT A MINUTE. I'm going to explain something to you.

When working with source code projects there are some files that you don't want to add to the repository, since they don't contribute anything. In the repository, as a general rule, there should be no executable files, no bytecode, no object code, and often no .zip, .rar, .jar, .war, etc. These files bloat the repository and, when we have many commits, they make the repository too large and can also slow down the download and upload work.

For each language and for each development environment it is recommended not to include certain types of files. These are the **files to ignore**. Each programmer can add or remove from the list those they consider appropriate. The files and folders to ignore must be specified in the `.gitignore` file. On each line you put a file, a folder or a regular expression indicating various types of files or folders.

> You can see that when creating our project in NetBeans the IDE has automatically generated a default `.gitignore` that has appeared next to the new `TestsGit` folder

in the repository https://github.com/github/gitignore you have many examples for different languages, build tools and environments.

For the Java language: https://github.com/github/gitignore/blob/master/Java.gitignore

For the Gradle tool: https://github.com/github/gitignore/blob/master/Gradle.gitignore

For the Netbeans environment: https://github.com/github/gitignore/blob/master/Global/NetBeans.gitignore

We, following the indications of this last link, are going to ignore the suggested folders and files. So the `.gitignore` file should have the following content:

```
**/nbproject/private/
**/nbproject/Makefile-*.mk
**/nbproject/Package-*.bash
build/
nbbuild/
dist/
nbdist/
.nb-gradle/
```

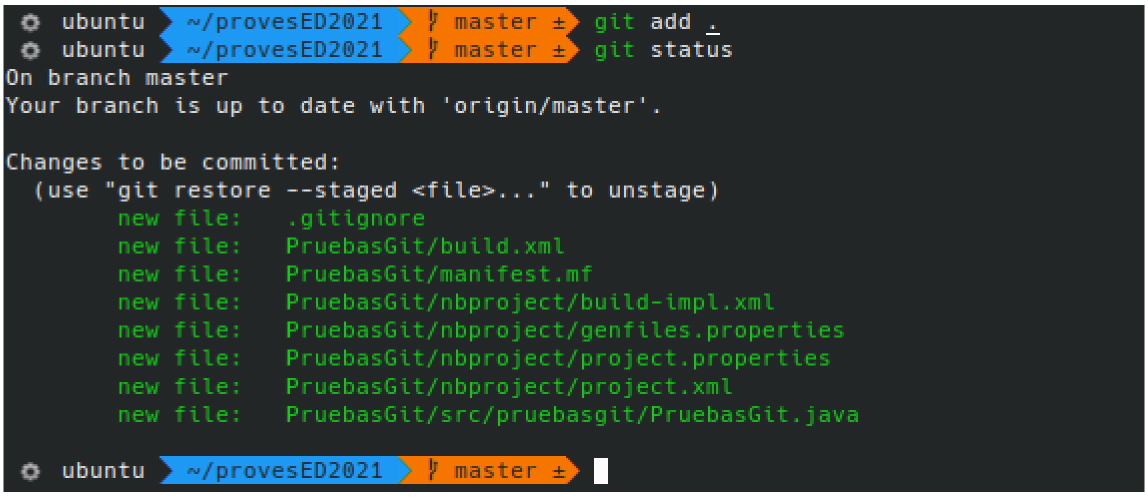
The trailing slash is optional, but I like to put it when referring to folders, so I know when it's a file and when it's a folder.

Create the file `.gitignore` with said content and take a screenshot.

Now yes, we do:

```
git add .
git status
```

We will see that the `dist`, `build` and `nbproject/private` folders do not appear, nor do any of the omitted files in `.gitignore`.



```
 ubuntu  ~/provesED2021  master ±  git add .
 ubuntu  ~/provesED2021  master ±  git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   PruebasGit/build.xml
        new file:   PruebasGit/manifest.mf
        new file:   PruebasGit/nbproject/build-impl.xml
        new file:   PruebasGit/nbproject/genfiles.properties
        new file:   PruebasGit/nbproject/project.properties
        new file:   PruebasGit/nbproject/project.xml
        new file:   PruebasGit/src/pruebasgit/PruebasGit.java

 ubuntu  ~/provesED2021  master ± 
```

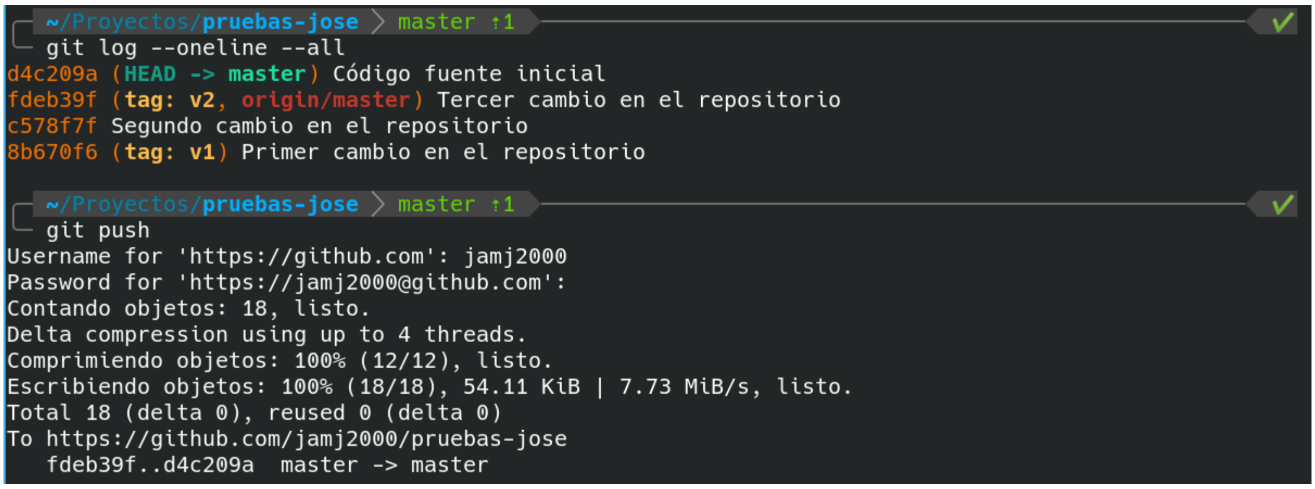
Now we can run

```
git commit -m "Código fuente inicial"
```

Notice that I have written `git add .`. The dot indicates the current directory, and is a way of indicating that I include in the staging area all the files in the directory I am in (except the files and folders indicated in `.gitignore`) This form of git is used quite a lot add when we don't want to add the files one by one.

7.3. Push changes from local repository to remote repository

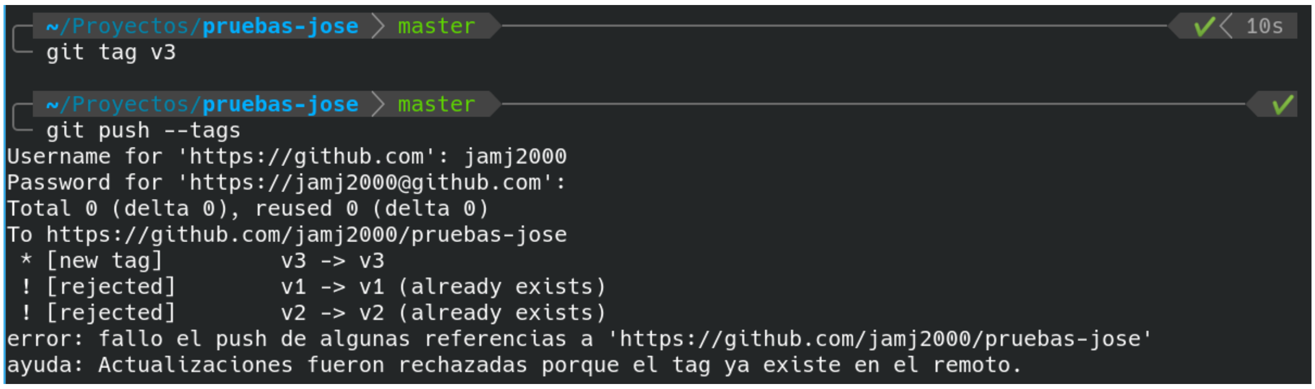
Now we only have to upload the changes made to the remote repository with **git push**



```
~/Proyectos/pruebas-jose  master ↑1                                        ✓
git log --oneline --all
d4c209a (HEAD -> master) Código fuente inicial
fdeb39f (tag: v2, origin/master) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio

~/Proyectos/pruebas-jose  master ↑1                                        ✓
git push
Username for 'https://github.com': jamj2000
Password for 'https://jamj2000@github.com':
Contando objetos: 18, listo.
Delta compression using up to 4 threads.
Comprimiendo objetos: 100% (12/12), listo.
Escribiendo objetos: 100% (18/18), 54.11 KiB | 7.73 MiB/s, listo.
Total 18 (delta 0), reused 0 (delta 0)
To https://github.com/jamj2000/pruebas-jose
   fdeb39f..d4c209a  master -> master
```

To make this section more interesting, we are going to create a tag in the current commit and upload it to github so that it creates a new *release*.

```
git tag v3
git push --tags
```



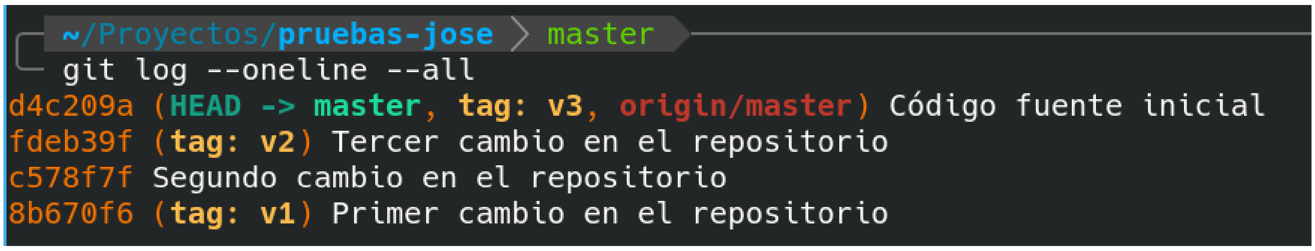
```
~/Proyectos/pruebas-jose > master                                    ✓< 10s
git tag v3

~/Proyectos/pruebas-jose > master                                          ✓
git push --tags
Username for 'https://github.com': jamj2000
Password for 'https://jamj2000@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/jamj2000/pruebas-jose
 * [new tag]         v3 -> v3
 ! [rejected]        v1 -> v1 (already exists)
 ! [rejected]        v2 -> v2 (already exists)
error: fallo el push de algunas referencias a 'https://github.com/jamj2000/pruebas-jose'
ayuda: Actualizaciones fueron rechazadas porque el tag ya existe en el remoto.
```

Well, the v1 and v2 tags are not uploaded because we had already uploaded them previously.

In this case, we could also have executed

```
git push origin v3
```

And the history of our local repository would be this beautiful



```
~/Proyectos/pruebas-jose > master
git log --oneline --all
d4c209a (HEAD -> master, tag: v3, origin/master) Código fuente inicial
fdeb39f (tag: v2) Tercer cambio en el repositorio
c578f7f Segundo cambio en el repositorio
8b670f6 (tag: v1) Primer cambio en el repositorio
```

Access your repository on GitHub and take a screenshot of the *Tags*.

Take another screenshot of the code files and folders uploaded to GitHub. Neither the `dist`, `build` folder, nor the `nbproject/private` folder should appear. And yes the `.gitignore` file should appear.

NOTE: The `.git` folder is never displayed on GitHub.

NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.

Upload a PDF document to the AULES platform with the relevant screenshots and explanations.

8. Using an SSH key pair

In our case (December 2021), gitHub no longer allows HTTP connections, only SSH, and we already did this at the beginning of the exercises, so you can skip step 8 and go directly to point 9. I'll leave it here as reference and consultation.

As you may have noticed, every time we do a `git push` it asks for the username and password. This is quite annoying.

One way around this is by using an **SSH key pair** (a private key and a public key). Both complement each other. One without the other is useless.

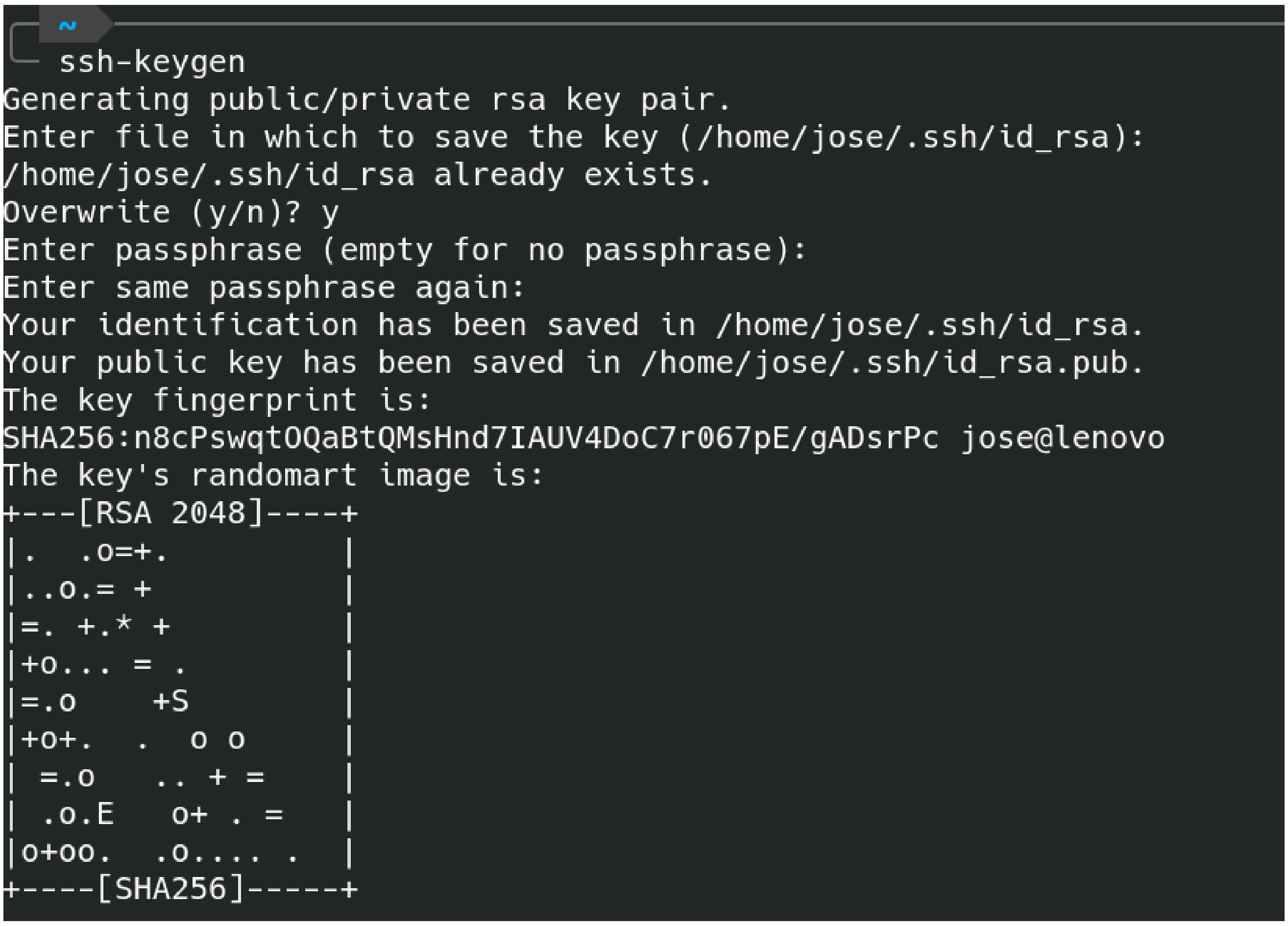
This method prevents our GitHub username and password from being saved to a disk file. Therefore it is very safe. In the event that someone logs in to our PC, they could access our passwords. In that case we would delete the key pair and re-create new ones and our GitHub username and password would never be compromised.

We will follow the following steps:

8.1. We generate a pair of SSH keys

It is very simple. As a normal user (without being root) we execute the command

```
ssh-keygen
```



```
~
 ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jose/.ssh/id_rsa):
/home/jose/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jose/.ssh/id_rsa.
Your public key has been saved in /home/jose/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:n8cPswqtOQaBtQMsHnd7IAUV4DoC7r067pE/gADsrPc jose@lenovo
The key's randomart image is:
+---[RSA 2048]----+
|.  .o=+.         |
|..o.= +          |
|=. +.* +         |
|+o... = .        |
|=.o    +S        |
|+o+.  .  o o     |
| =.o   .. + =    |
| .o.E   o+ . =   |
|o+oo.  .o.... .  |
+----[SHA256]-----+
```

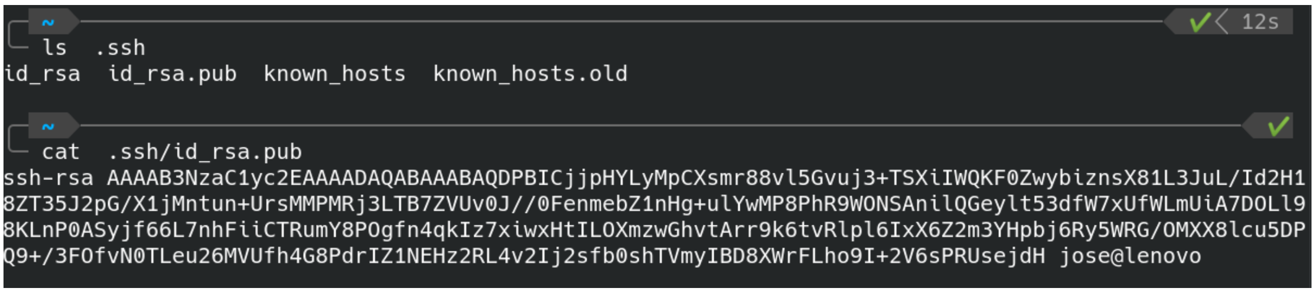
We press Enter to everything. Unless a previous key pair already exists. In that case it will ask us if we want to override (Override (y/n)?) In this case, in this question we answer y . Then all Enter.

This will create a **~/.ssh** folder and inside at least 2 files:

- **id_rsa**
- **id_rsa.pub**

The first file corresponds to the private key and the second to the public key.

We copy the content of the public key into a text editor. We will need it later.



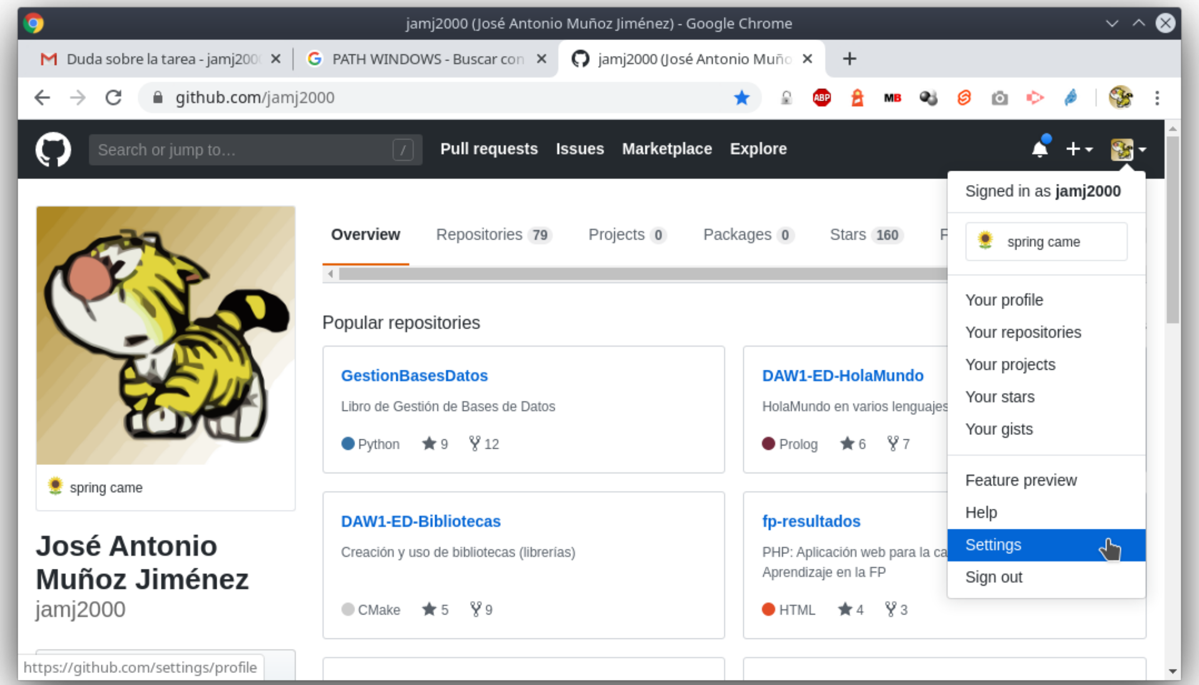
```
 ~                                                                    ✓ 12s
 ls  .ssh
id_rsa  id_rsa.pub  known_hosts  known_hosts.old

 ~                                                                        ✓
 cat  .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDPBICjjpHYLyMpCXsmr88vl5Gvuj3+TSXiIWQKF0ZwybiznsX81L3JuL/Id2H1
8ZT35J2pG/X1jMntun+UrsMMPMRj3LTB7ZVUv0J//0FenmebZ1nHg+ulYwMP8PhR9WONSAnilQGeylt53dfW7xUfWLmUiA7DOLl9
8KLnP0ASyjf66L7nhFiiCTRumY8POgfn4qkIz7xiwxHtILOXmzwGhvtArr9k6tvRlpl6IxX6Z2m3YHpbj6Ry5WRG/OMXX8lcu5DP
Q9+/3FOfvN0TLeu26MVUfh4G8PdrIZ1NEHz2RL4v2Ij2sfb0shTVmyIBD8XWrFLho9I+2V6sPRUsejdH jose@lenovo
```

ssh-rsa must be copied .... jose@lenovo

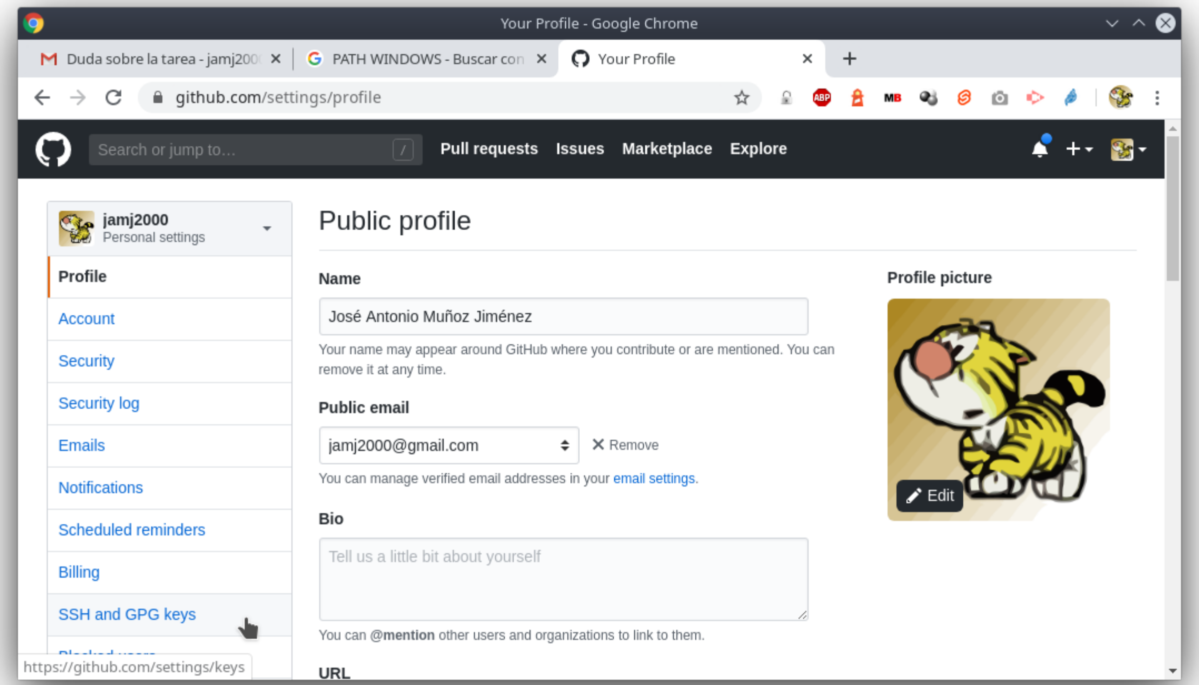
In your case, instead of jose@lenovo another user and pc will appear.

8.2. We add public ssh key to github.

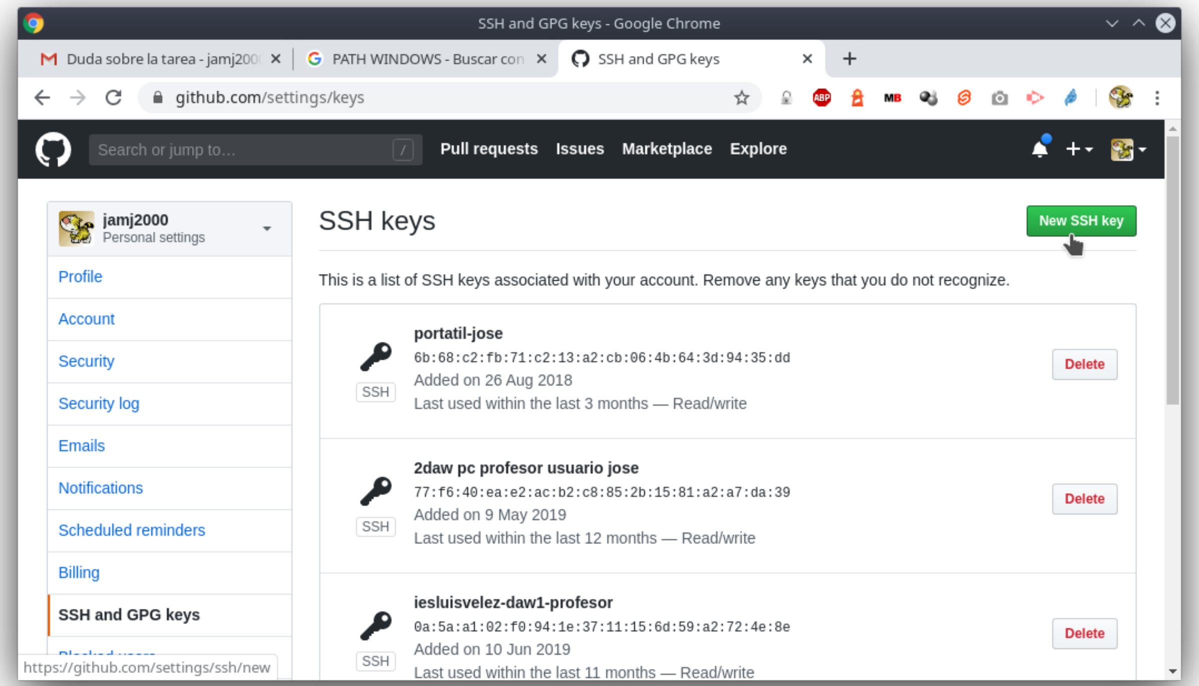
We log in to GitHub and in the general menu (upper right corner) we select the **Settings** option.



Then, on the left, we choose the option **SSH and GPG keys**


Your Profile - Google Chrome
Duda sobre la tarea - jamj200
PATH WINDOWS - Buscar con
Your Profile
github.com/settings/profile
Search or jump to...
Pull requests
Issues
Marketplace
Explore
jamj2000
Personal settings
Profile
Account
Security
Security log
Emails
Notifications
Scheduled reminders
Billing
SSH and GPG keys
https://github.com/settings/keys
Public profile
Name
José Antonio Muñoz Jiménez
Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.
Public email
jamj2000@gmail.com
Remove
You can manage verified email addresses in your email settings.
Bio
Tell us a little bit about yourself
You can @mention other users and organizations to link to them.
URL
Profile picture
Edit

Next, on the right, click on the button **New SSH key**



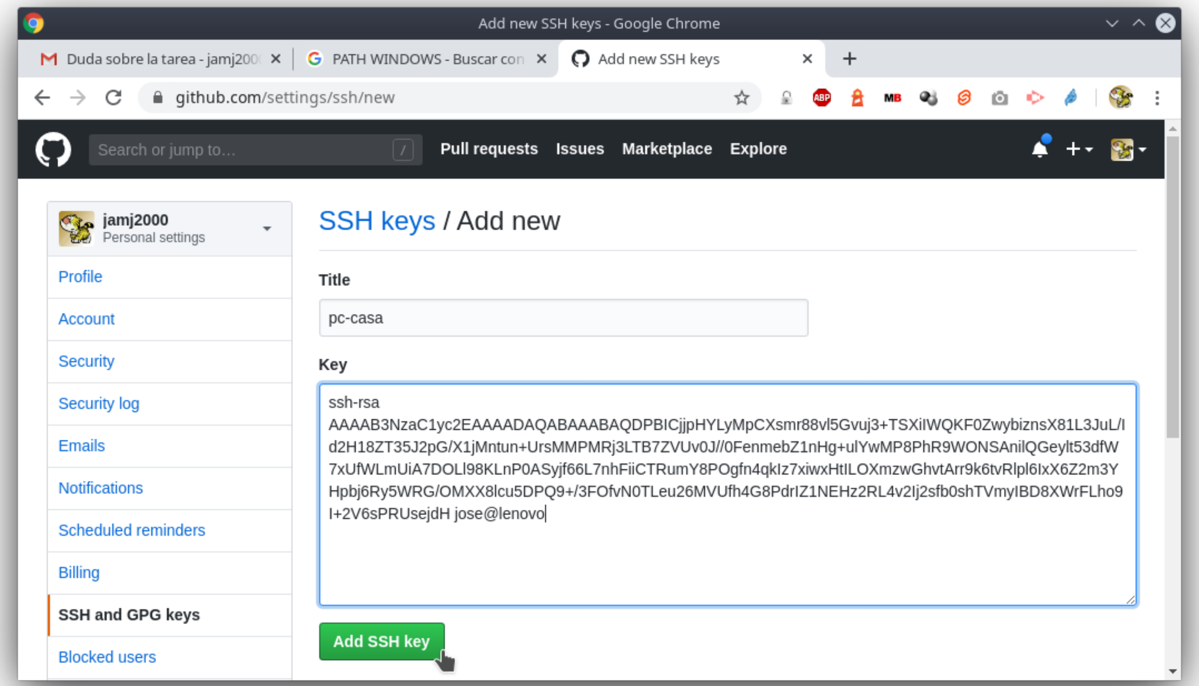
This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

portatil-jose
```
6b:68:c2:fb:71:c2:13:a2:cb:06:4b:64:3d:94:35:dd
```
Added on 26 Aug 2018
Last used within the last 3 months — Read/write

2daw pc profesor usuario jose
```
77:f6:40:ea:e2:ac:b2:c8:85:2b:15:81:a2:a7:da:39
```
Added on 9 May 2019
Last used within the last 12 months — Read/write

iesluisvelez-daw1-profesor
```
0a:5a:a1:02:f0:94:1e:37:11:15:6d:59:a2:72:4e:8e
```
Added on 10 Jun 2019
Last used within the last 11 months — Read/write

Then we put a name to the key, for example pc-house. And we copy the content of the public key. Finally, click on the button **Add SSH key**



Title

pc-casa

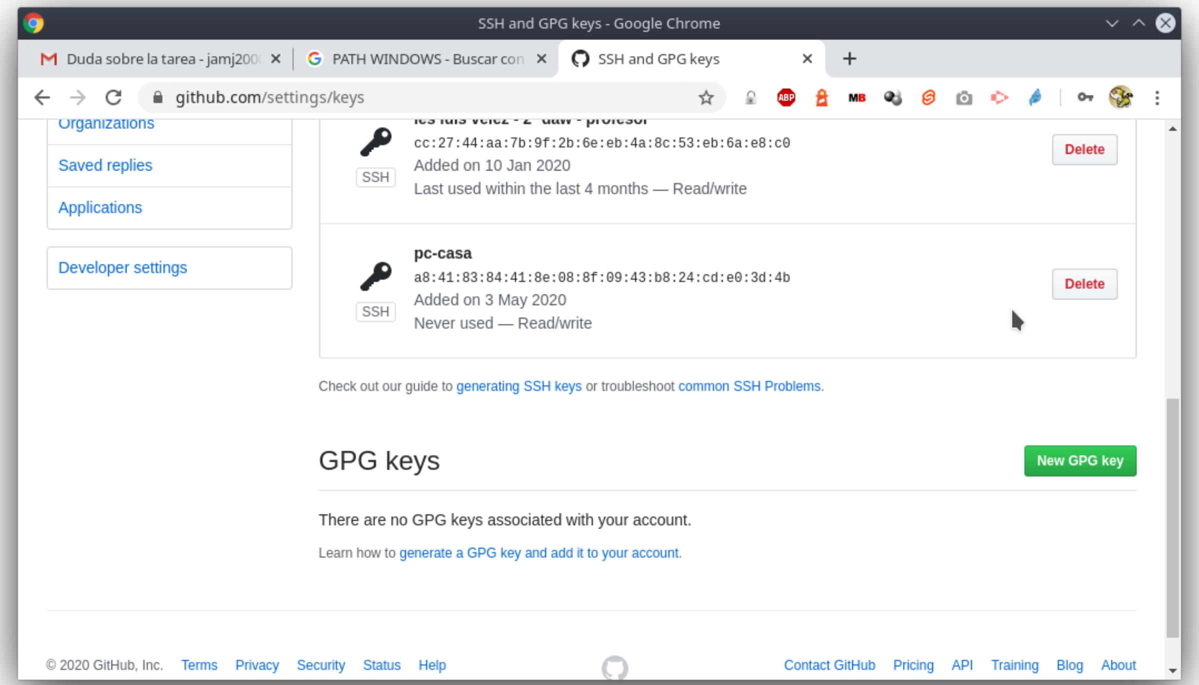
Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDPBICjjpHYLyMpCXsmr88vl5Gvuj3+TSXiIWQKF0ZwybiznsX81L3JuL/I
d2H18ZT35J2pG/X1jMntun+UrsMMPMRj3LTB7ZVUv0J//0FenmebZ1nHg+ulYwMP8PhR9WONSAnilQGeylt53dfW
7xUfWLmUiA7DOLl98KLnP0ASyjf66L7nhFiiCTRumY8POgfn4qkIz7xiwxHtILOXmzwGhvtArr9k6tvRlpl6IxX6Z2m3Y
Hpbj6Ry5WRG/OMXX8lcu5DPQ9+/3FOfvN0TLeu26MVUfh4G8PdrIZ1NEHz2RL4v2Ij2sfb0shTVmyIBD8XWrFLho9
I+2V6sPRUsejdH jose@lenovo
```

The above key can be used for any of our repositories. To make use of it, all we need is the URL in SSH format of each repository.

8.3. We check that it has been created correctly

If, for any reason, someone accessed our PC and took the private key, it would be enough to delete this public key from GitHub and the thief would not use our private key at all.


SSH and GPG keys - Google Chrome

github.com/settings/keys

Organizations
Saved replies
Applications
Developer settings

ies luis velez - 2 - daw - profesor
cc:27:44:aa:7b:9f:2b:6e:eb:4a:8c:53:eb:6a:e8:c0
Added on 10 Jan 2020
Last used within the last 4 months — Read/write

pc-casa
a8:41:83:84:41:8e:08:8f:09:43:b8:24:cd:e0:3d:4b
Added on 3 May 2020
Never used — Read/write

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.

GPG keys

There are no GPG keys associated with your account.

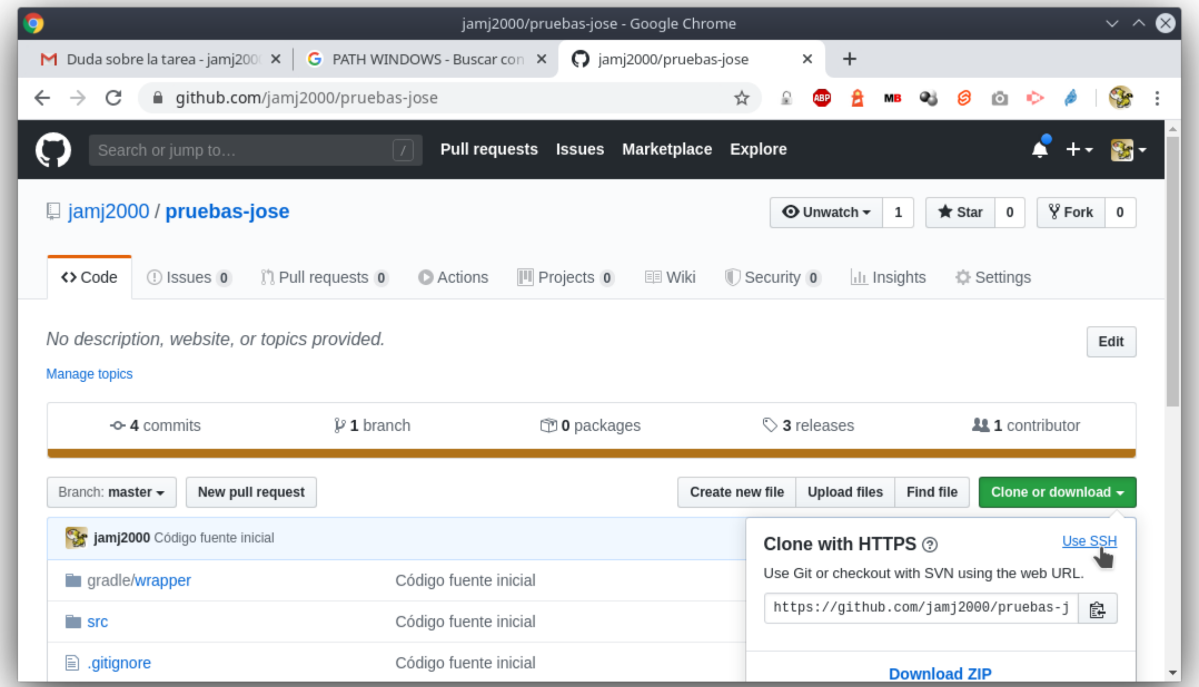
Learn how to generate a GPG key and add it to your account.

© 2020 GitHub, Inc.

8.4. Getting repository SSH URL

Clone or download button, **Use SSH**

We copy URL in SSH format. Its format is relatively easy to memorize. Always git@githbub.com followed by a colon : and then the username / repository name.


jamj2000/pruebas-jose - Google Chrome

github.com/jamj2000/pruebas-jose

jamj2000 / pruebas-jose

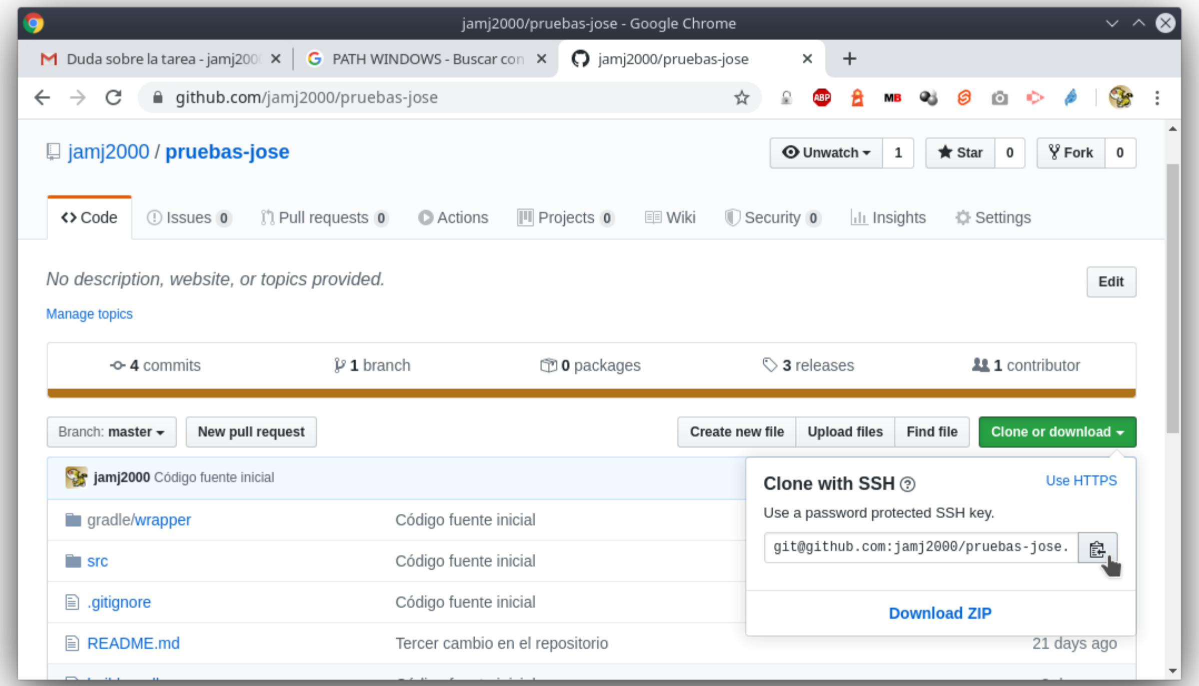
No description, website, or topics provided.

4 commits · 1 branch · 0 packages · 3 releases · 1 contributor

Branch: master

jamj2000 Código fuente inicial	
gradle/wrapper	Código fuente inicial
src	Código fuente inicial
.gitignore	Código fuente inicial

Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/jamj2000/pruebas-j



No description, website, or topics provided.

jamj2000 Código fuente inicial

Use a password protected SSH key.

```
git@github.com:jamj2000/pruebas-jose.
```

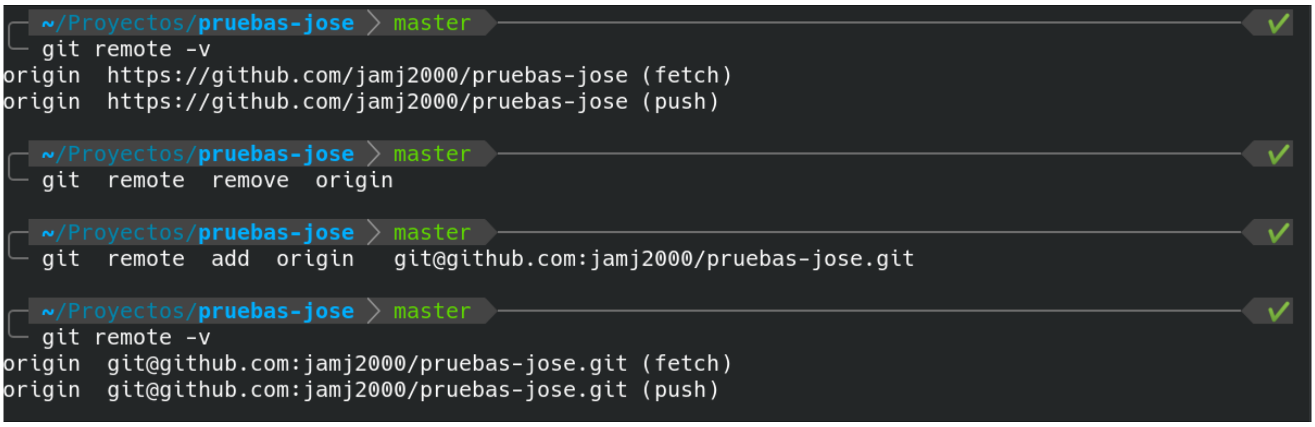
gradle/wrapper	Código fuente inicial	
src	Código fuente inicial	
.gitignore	Código fuente inicial	
README.md	Tercer cambio en el repositorio	21 days ago

8.5. Associating our local repository via SSH

Our local repository was associated with origin via HTTPS. We must unsubscribe said link and create a new one that makes use of the SSH protocol.

We run

```
git  remote  remove  origin
git  remote  add  origin   git@github.com:your_user/your_repository
```



```
~/Proyectos/pruebas-jose  master
git remote -v
origin  https://github.com/jamj2000/pruebas-jose (fetch)
origin  https://github.com/jamj2000/pruebas-jose (push)

~/Proyectos/pruebas-jose  master
git  remote  remove  origin

~/Proyectos/pruebas-jose  master
git  remote  add  origin   git@github.com:jamj2000/pruebas-jose.git

~/Proyectos/pruebas-jose  master
git remote -v
origin  git@github.com:jamj2000/pruebas-jose.git (fetch)
origin  git@github.com:jamj2000/pruebas-jose.git (push)
```

8.6. We create a commit and push it to GitHub.

To verify that we are not prompted for a username and password when we do a git push, we are going to modify the README.md file, create a commit, and upload it to GitHub.

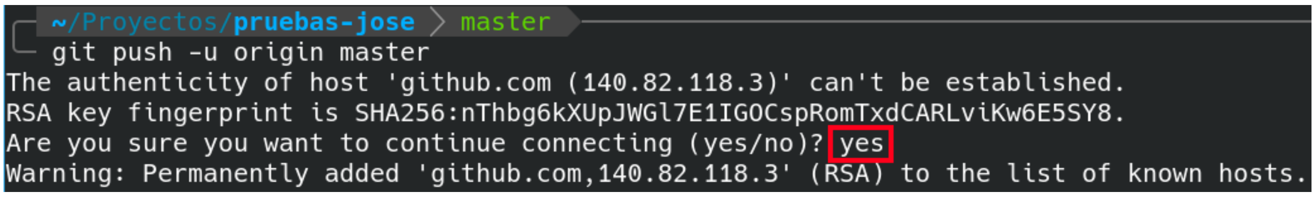
We will put at the beginning of each line the symbol > and a space. The README.md file would look something like this:

```
> David Martinez
> 12 November 2021
> IES Mestre Ramón Esteve
```

Then we save and execute

```
git add README.md
git commit -m "Added appointment"
git push -u origin master
```

Running the last command will make an SSH connection to GitHub.



```
~/Proyectos/pruebas-jose  master
git push -u origin master
The authenticity of host 'github.com (140.82.118.3)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,140.82.118.3' (RSA) to the list of known hosts.
```

When an SSH connection is made with a new key, confirmation is requested the first time and you must type **yes**.
After that, the remote host will be registered in the **.ssh/known_hosts** file. Subsequent times confirmation is no longer required, as long as the `.ssh/known_hosts` file contains such records.

> *NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.*

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

9. Resolving conflicts

In this activity we will see what is meant by conflict, when it occurs and how to resolve it.

As you know, the same repository can have copies in different places. Right now we have a copy on GitHub and a local one on our PC. But there could be more local copies on other PCs.

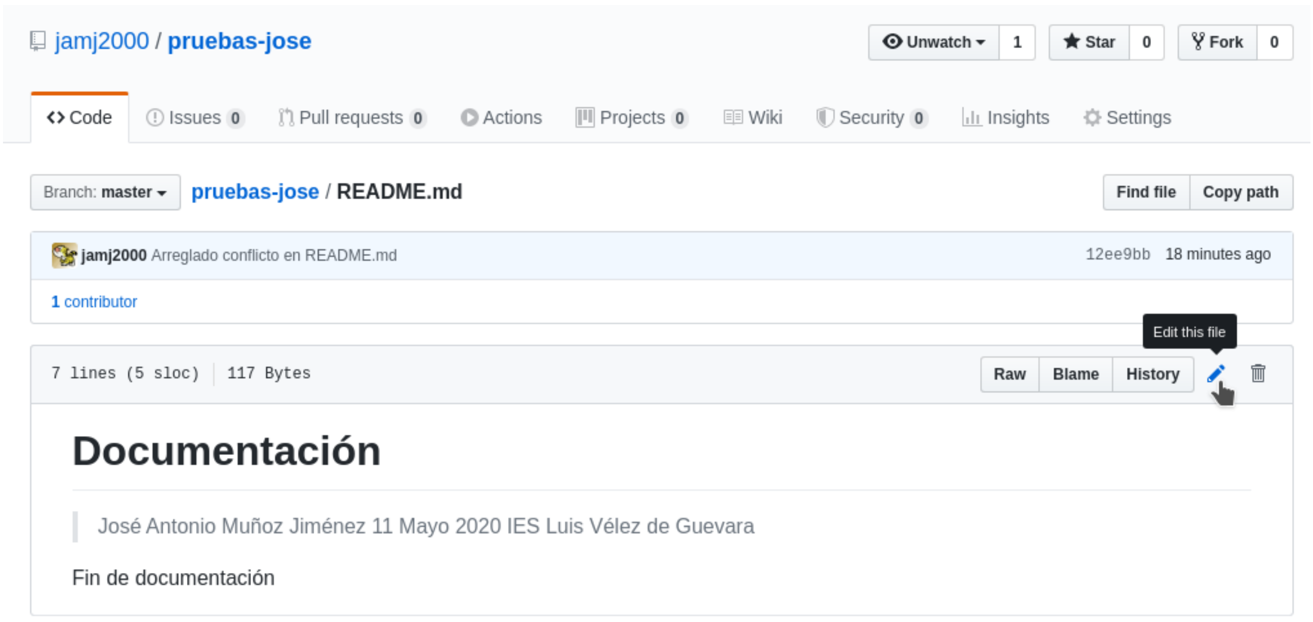
Whenever we make changes (ie commits) to the same file on the same lines but on different copies, a conflict will occur.

To see this, we're going to commit to our repository on GitHub, and then commit to our local repository. We will work with the `README.md` file only.

9.1. We modify remote README.md file

On GitHub we are going to modify the `README.md` file and register the change (commit).

To do this, we enter our remote repository, click on the `README.md` file and then click on the pencil to edit.



pruebas-jose / README.md

jamj2000 Arreglado conflicto en README.md 12ee9bb 18 minutes ago

1 contributor

7 lines (5 sloc) | 117 Bytes

Documentación

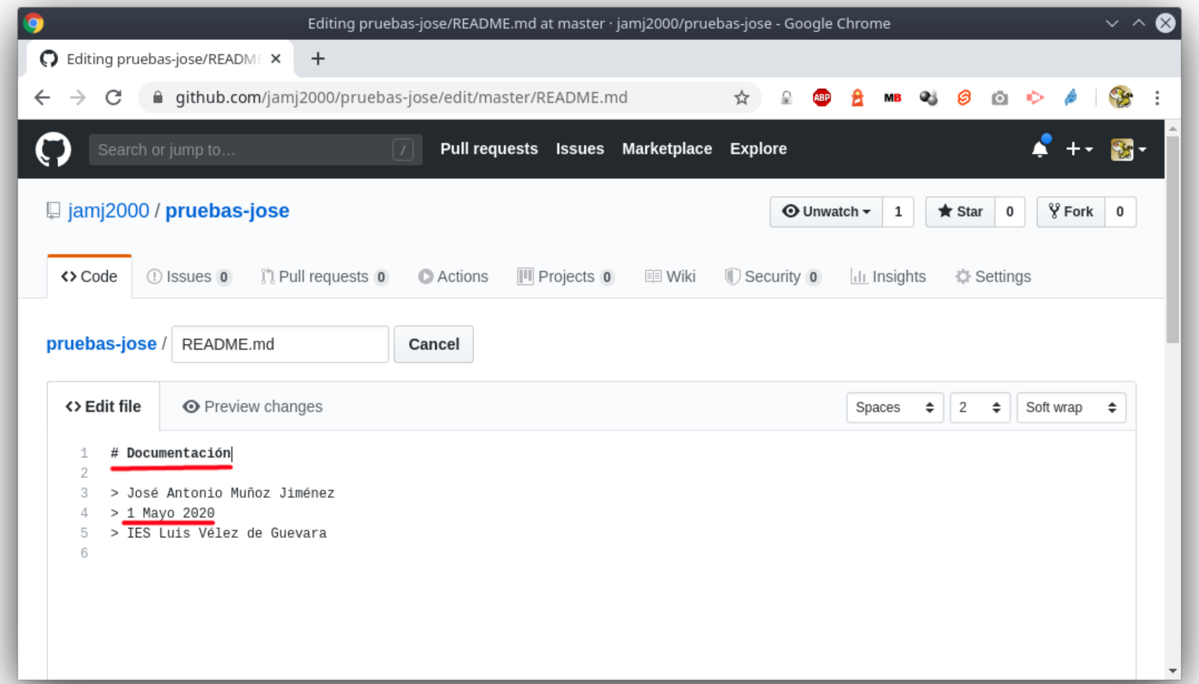
> José Antonio Muñoz Jiménez 11 Mayo 2020 IES Luis Vélez de Guevara

Fin de documentación

Recently (mid-August 2021) gitHub added an interesting functionality to all of its repositories, and that is the ability to open the online vsCode editor for any repository simply by using the "`.`" `hotkey`.

Therefore we can make this modification as shown in the screenshots, or press the "`.`" (period) and use vsCode Online to make the modification.

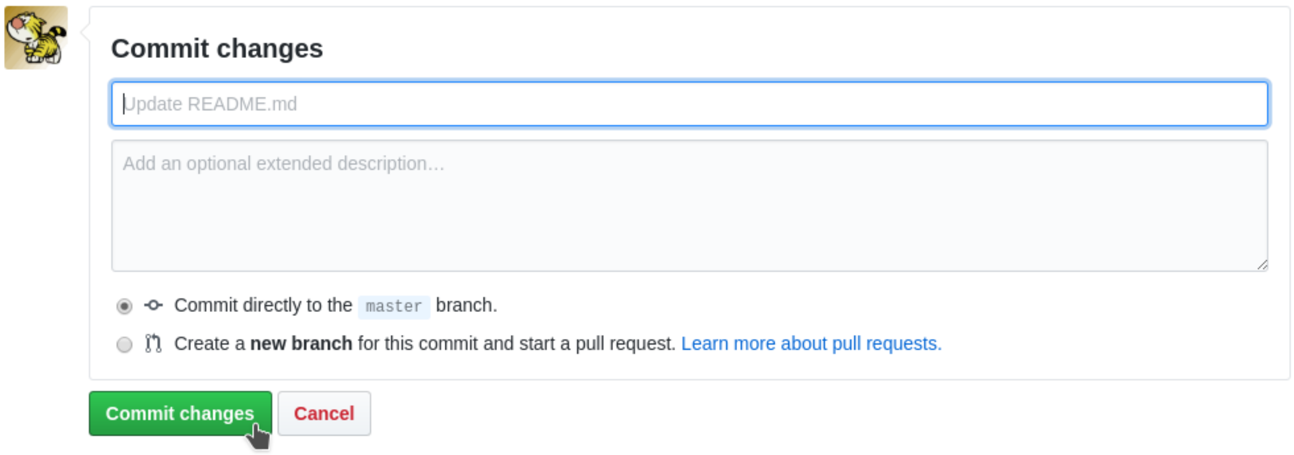
We insert a first line with title # and modify the date line.



```
# Documentación

> José Antonio Muñoz Jiménez
> 1 Mayo 2020
> IES Luis Vélez de Guevara
```

We register the commit. To do this, click on **Commit changes**

If we wish, we can put a message to the commit and a description, although it is not mandatory. GitHub sets one by default.

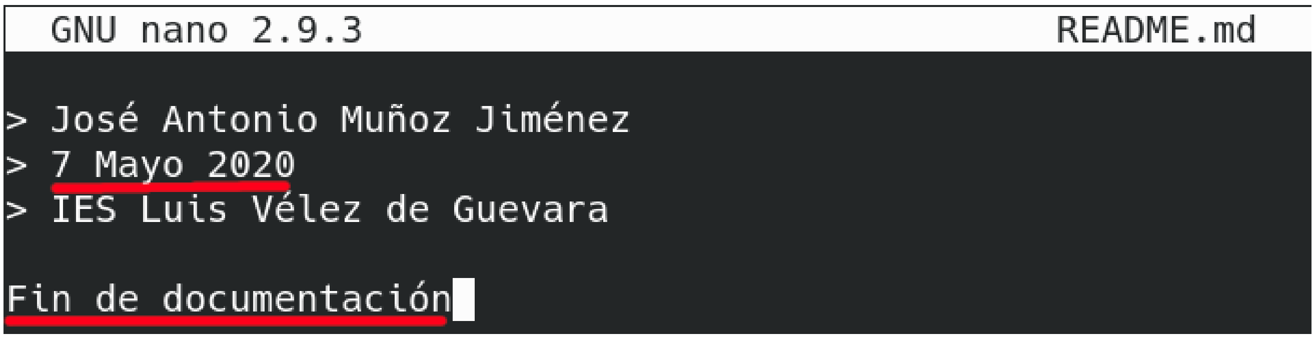

Commit changes
Update README.md
Add an optional extended description…
Commit directly to the master branch.
Create a new branch for this commit and start a pull request. Learn more about pull requests.
Commit changes
Cancel

9.2. We modify local README.md file

In our local repository, we are also going to modify the README.md file.

In this case we will add a line to the end of the file and modify the date line.

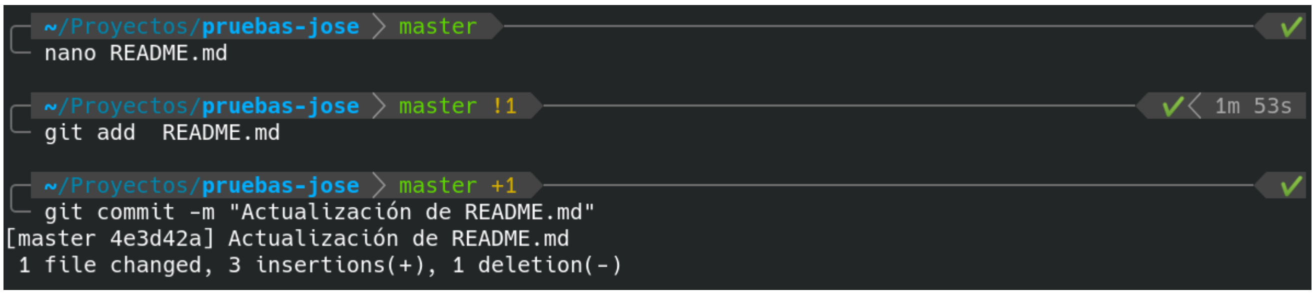
```
nano  README.md
```



```
GNU nano 2.9.3                                           README.md

> José Antonio Muñoz Jiménez
> 7 Mayo 2020
> IES Luis Vélez de Guevara

Fin de documentación
```

We save the changes and record commit.

```
git add README.md
git commit -m "README.md update"
```

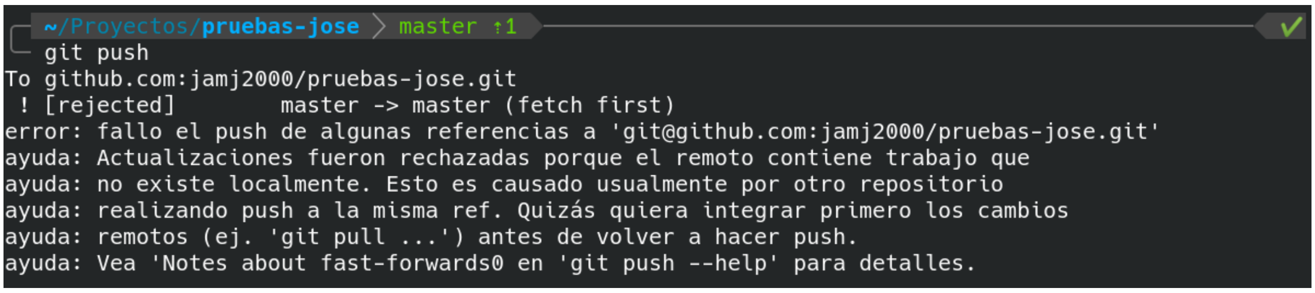


```
~/Proyectos/pruebas-jose  master
nano README.md

~/Proyectos/pruebas-jose  master !1                         ✓ 1m 53s
git add  README.md

~/Proyectos/pruebas-jose  master +1
git commit -m "Actualización de README.md"
[master 4e3d42a] Actualización de README.md
 1 file changed, 3 insertions(+), 1 deletion(-)
```

9.3. 3. We try to upload the local commit

Attempting to push our local commit to the remote repository will reject it.

```
git  push
```



```
~/Proyectos/pruebas-jose  master ↑1
git push
To github.com:jamj2000/pruebas-jose.git
 ! [rejected]        master -> master (fetch first)
error: fallo el push de algunas referencias a 'git@github.com:jamj2000/pruebas-jose.git'
ayuda: Actualizaciones fueron rechazadas porque el remoto contiene trabajo que
ayuda: no existe localmente. Esto es causado usualmente por otro repositorio
ayuda: realizando push a la misma ref. Quizás quiera integrar primero los cambios
ayuda: remotos (ej. 'git pull ...') antes de volver a hacer push.
ayuda: Vea 'Notes about fast-forwards0 en 'git push --help' para detalles.
```

This is not a conflict.
It simply tells us that we must first update our local repository with the content of the remote repository.

If we have made changes to our remote repository, we will need to integrate them into our local repository before we can upload new local changes.

9.4. Conflict occurs

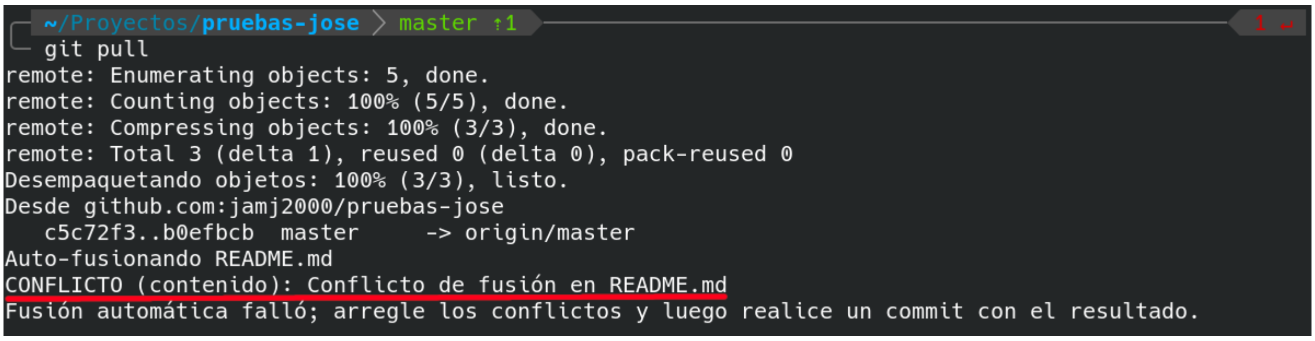
So we do

```
git  pull
```

to **download commits from the remote repository** that we don't have locally.

This should not cause a conflict.
But in this case it does occur, because we have modified the same file (`README.md`) and also on the same line (the date line)

So the merge is done, but it warns us that there is a conflict in that file. We will have to solve it manually.



```
~/Proyectos/pruebas-jose  master ↑1
 git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Desempaquetando objetos: 100% (3/3), listo.
Desde github.com:jamj2000/pruebas-jose
   c5c72f3..b0efbcb  master     -> origin/master
Auto-fusionando README.md
CONFLICTO (contenido): Conflicto de fusión en README.md
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
```

9.5. We fix the conflict

To fix the conflict, we open the file in question and in the line or lines where the conflict has occurred we will see some marks like the following:

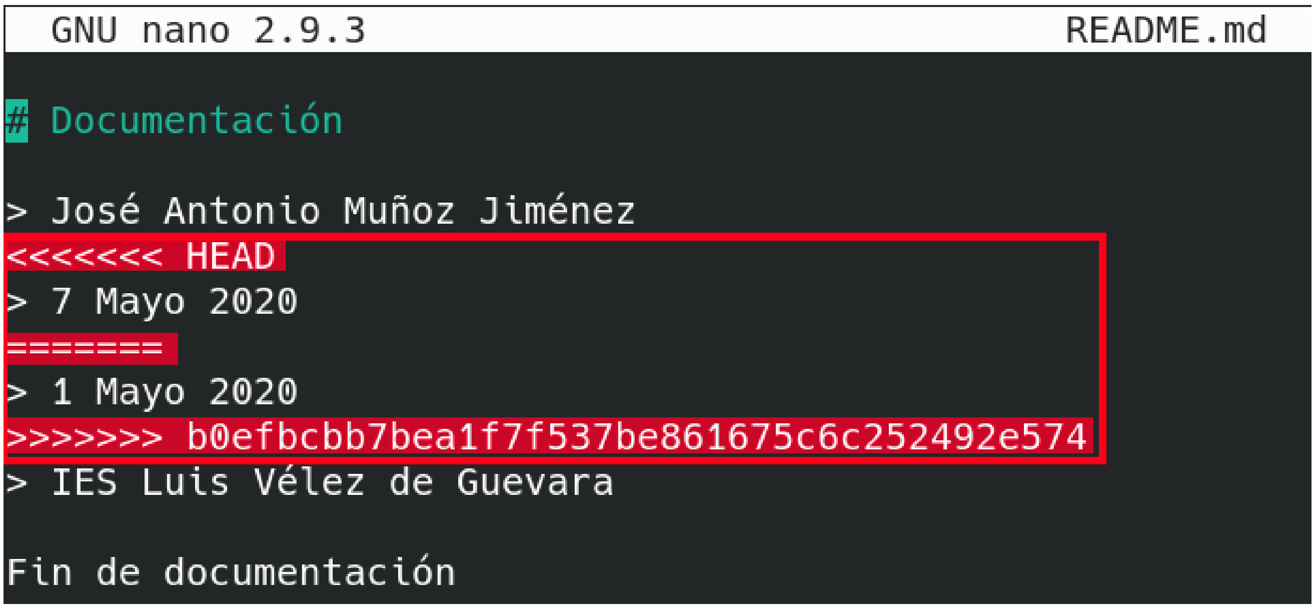
`<<<<<<<`

```
    line or lines in local commit
```

`=======`

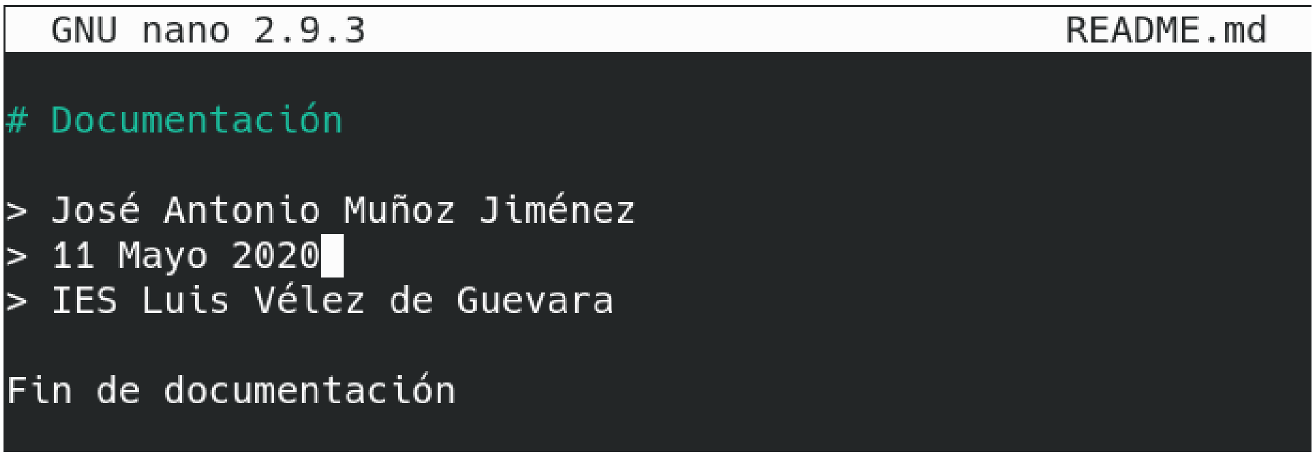
```
   line or lines in remote commit
```

`>>>>>>>`



```
  GNU nano 2.9.3                                       README.md

# Documentación

> José Antonio Muñoz Jiménez
<<<<<<< HEAD
> 7 Mayo 2020
=======
> 1 Mayo 2020
>>>>>>> b0efbcbb7bea1f7f537be861675c6c252492e574
> IES Luis Vélez de Guevara

Fin de documentación
```

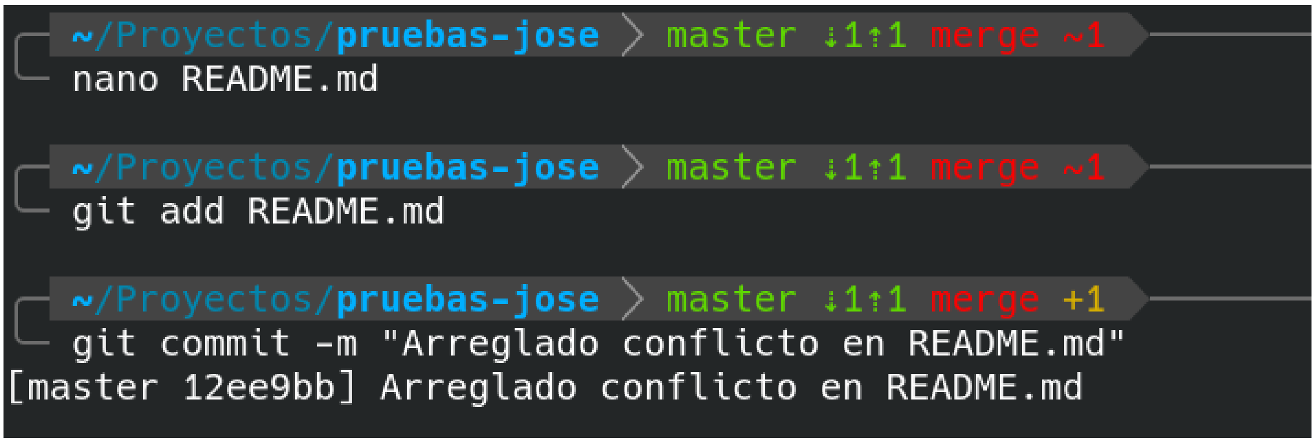
Resolving the conflict consists of choosing one of the 2 options and removing the previous marks.
Although we can also not choose any of the options and write another instead.
This is what I have done here by setting the date May 11.



```
GNU nano 2.9.3                                                    README.md

# Documentación

> José Antonio Muñoz Jiménez
> 11 Mayo 2020
> IES Luis Vélez de Guevara

Fin de documentación
```

Next, we save the changes. And we register a new commit.

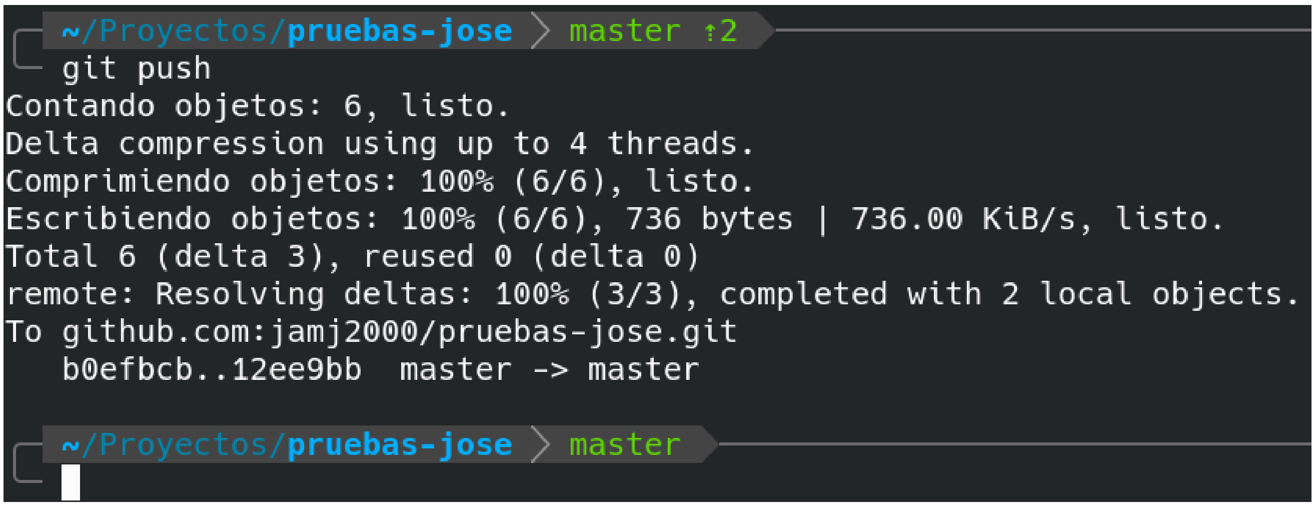
```
git add README.md
git commit  -m "Fixed conflict in README.md"
```



```
~/Proyectos/pruebas-jose > master ↓1↑1 merge ~1
nano README.md

~/Proyectos/pruebas-jose > master ↓1↑1 merge ~1
git add README.md

~/Proyectos/pruebas-jose > master ↓1↑1 merge +1
git commit -m "Arreglado conflicto en README.md"
[master 12ee9bb] Arreglado conflicto en README.md
```

Now we can upload our commit with the conflict resolved.

`git push`



```
~/Proyectos/pruebas-jose > master ↑2
git push
Contando objetos: 6, listo.
Delta compression using up to 4 threads.
Comprimiendo objetos: 100% (6/6), listo.
Escribiendo objetos: 100% (6/6), 736 bytes | 736.00 KiB/s, listo.
Total 6 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:jamj2000/pruebas-jose.git
   b0efbcb..12ee9bb  master -> master

~/Proyectos/pruebas-jose > master
```

NOTE: To avoid situations like the above, it is advisable not to make changes on GitHub, and if we have made them or uploaded commits from another local repository, the first thing we should do is `git pull`, resolve any conflicts that may arise, make the local commits that we want and finally upload commits to GitHub. In summary, a good strategy can be the following: at the beginning of the day we will do `git pull`, and at the end of the day we will do `git push`.

NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

10. Branch Creation

In this activity we are going to start working with branches. Specifically, we will see how to **create new branches**.

We can define a branch as a **parallel development within the same repository**. We can start such parallel development on any commit.

In essence, the main purposes of the branches are 2:

- **make changes to the repository without affecting the master branch**. Also applicable to other branches.
- **make changes in the repository and integrate them later in the master branch**. Also applicable to other branches

By default each git repository has a **master branch**. This is the main branch.
For security reasons, it is common to make the changes in some other branch and later integrate them into the master branch.
There are **workflows** (workflows)in which only commits are created in the master branch, only commits from other branches are integrated.

In this activity we will use 2 methods to work with new branches:

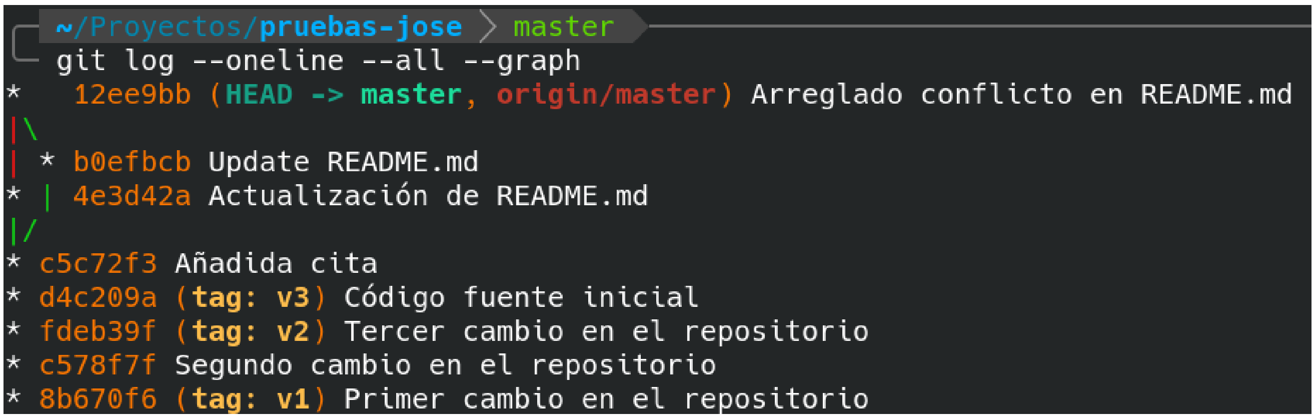
- `git checkout -b` *`new-branch`*
- `git branch` *`new-branch`* , and then `git checkout` *`new-branch`*

Let's check first, the current state of our repository. With `git log ...` we can see that we only have the master branch.

For this we execute

```
git log --oneline --all --graph
```

The option `--graph` allows us to view the branches "graphically".



```
~/Proyectos/pruebas-jose  master
git log --oneline --all --graph
*   12ee9bb (HEAD -> master, origin/master) Arreglado conflicto en README.md
|\
| * b0efbcb Update README.md
* | 4e3d42a Actualización de README.md
|/
* c5c72f3 Añadida cita
* d4c209a (tag: v3) Código fuente inicial
* fdeb39f (tag: v2) Tercer cambio en el repositorio
* c578f7f Segundo cambio en el repositorio
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

We can also see "another branch" without a name with the commit `b0ef` `Update README.md`.
This is actually the commit we edited on GitHub in a previous activity and had to merge into the local master branch, before pushing it back to GitHub.

10.1. Create branch using `git checkout -b ...`

The `git checkout -b new-branch` command has essentially 2 forms:

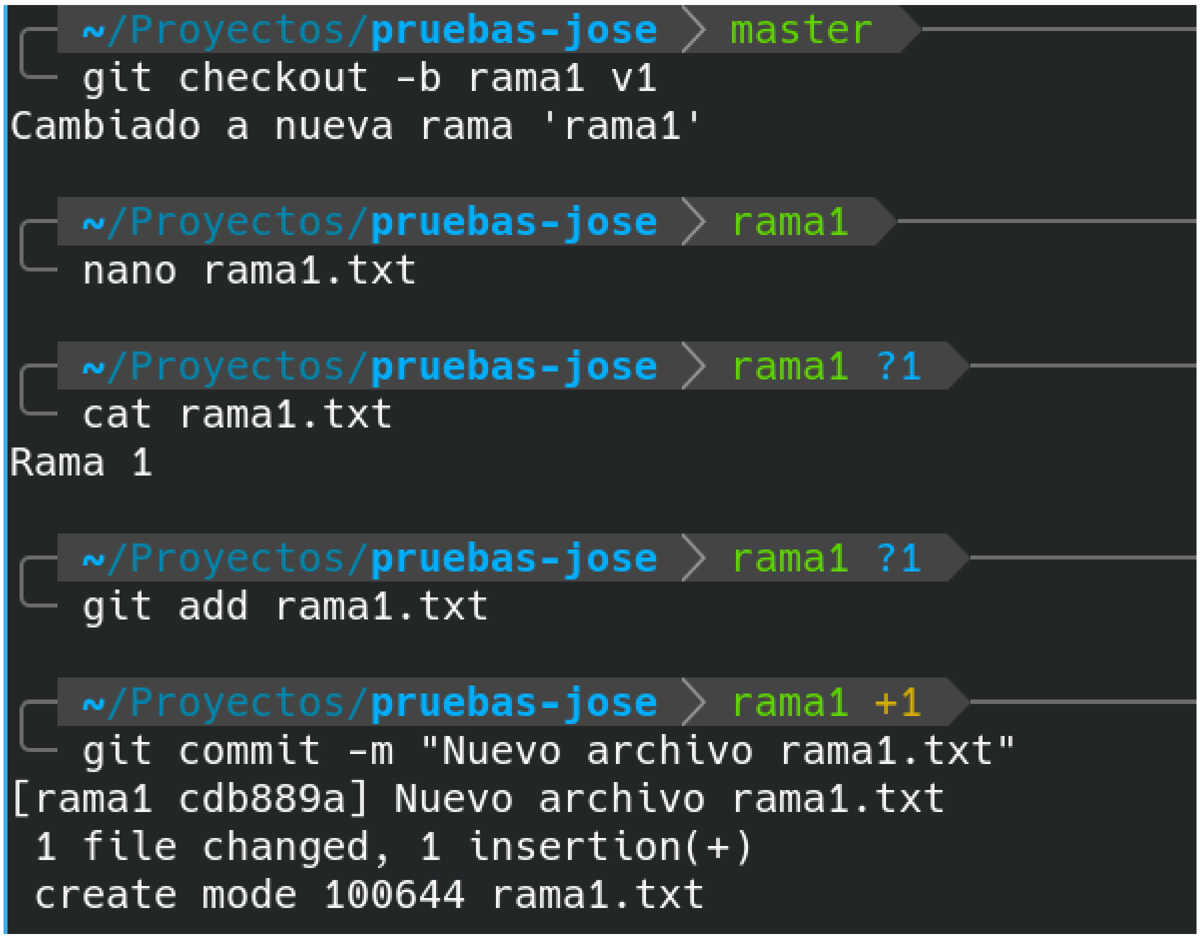
1. **`git checkout -b new-branch`** (Create a new branch from the current commit, and move to it)
2. **`git checkout -b new-branch commit-from-start`** (We create a new branch from the indicated commit, and move to it)

In this section we are going to create 2 branches (we will call them `branch1` and `branch2`) from the first commit,
i.e. the oldest commit, which we have labeled `v1`.

To create `branch1` and move to it, we will use the most direct way. For this we do:

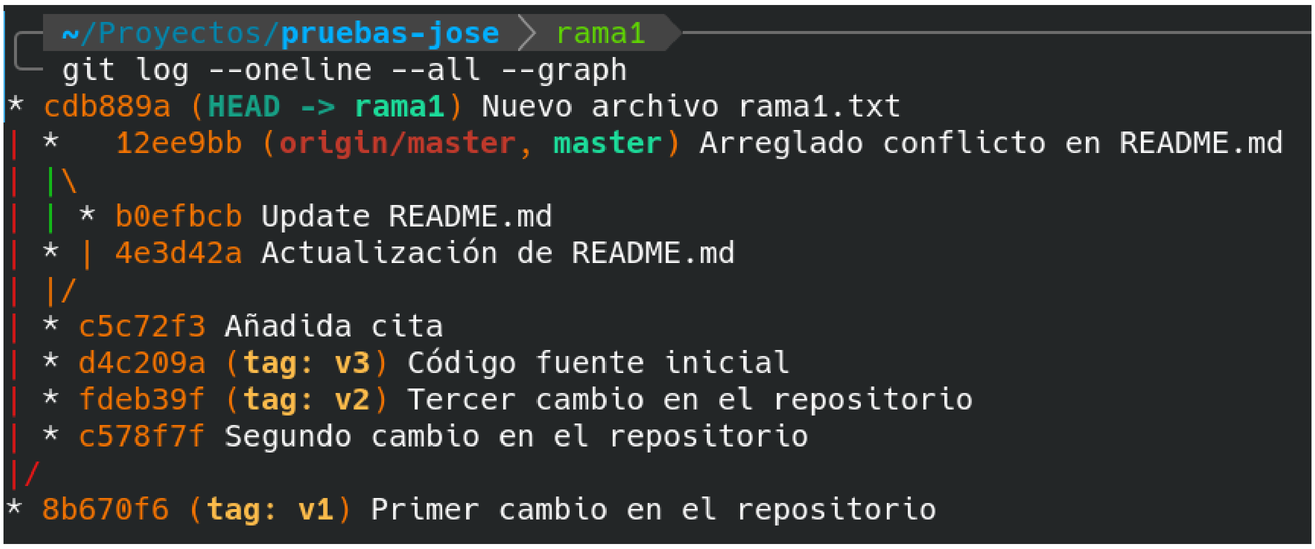
```
git checkout -b rama1 v1
```

On said `branch1`, we create a new commit.



```
~/Proyectos/pruebas-jose > master
git checkout -b rama1 v1
Cambiado a nueva rama 'rama1'

~/Proyectos/pruebas-jose > rama1
nano rama1.txt

~/Proyectos/pruebas-jose > rama1 ?1
cat rama1.txt
Rama 1

~/Proyectos/pruebas-jose > rama1 ?1
git add rama1.txt

~/Proyectos/pruebas-jose > rama1 +1
git commit -m "Nuevo archivo rama1.txt"
[rama1 cdb889a] Nuevo archivo rama1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 rama1.txt
```

The result is

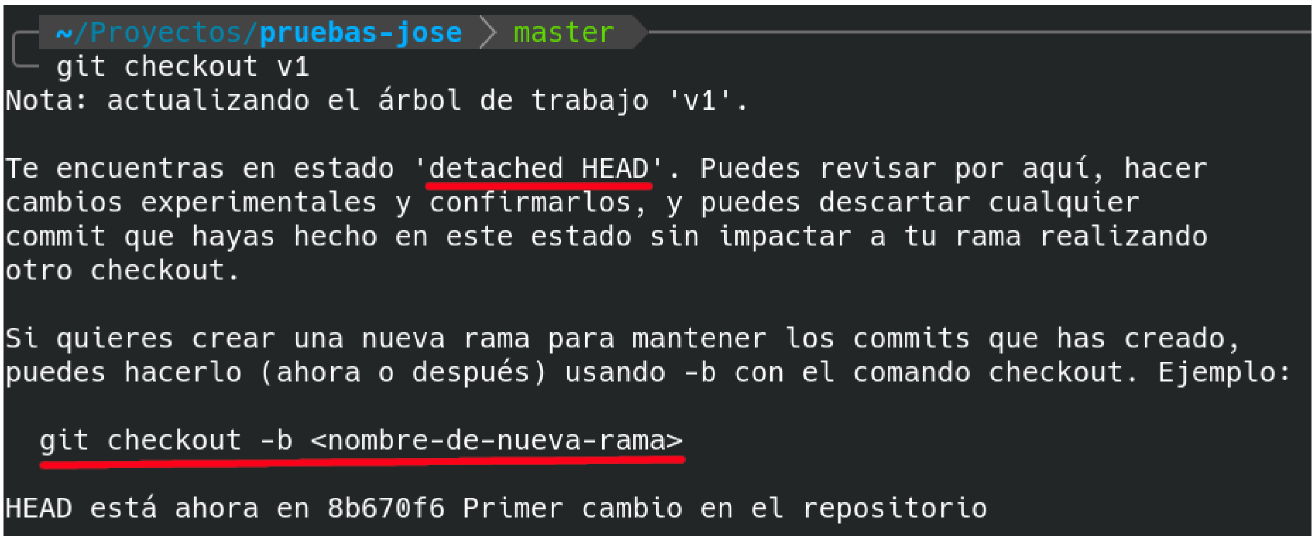


```
~/Proyectos/pruebas-jose > rama1
git log --oneline --all --graph
* cdb889a (HEAD -> rama1) Nuevo archivo rama1.txt
| *   12ee9bb (origin/master, master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

Now let's make another branch called `branch2` from commit `v1`, in a slightly different way.

Let's imagine that we've accidentally moved to the `v1` commit with

`git checkout v1`

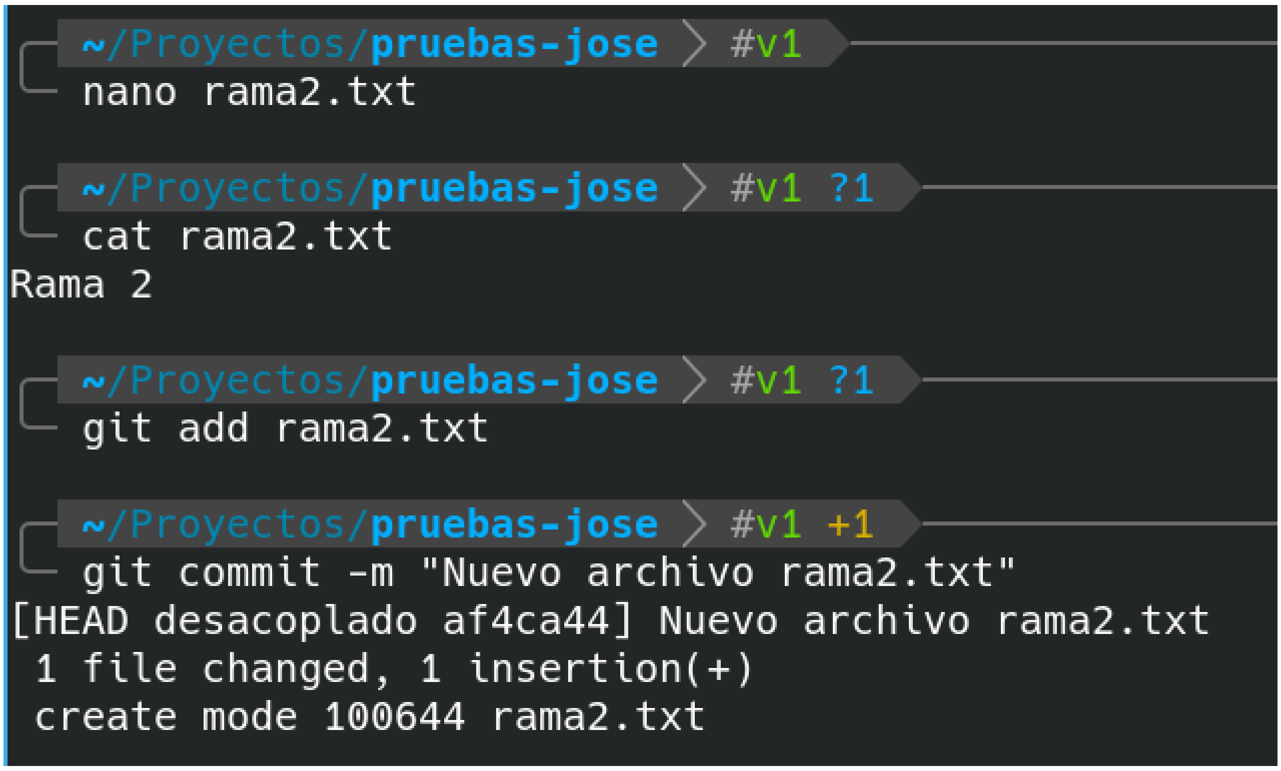


```
~/Proyectos/pruebas-jose > master
git checkout v1
Nota: actualizando el árbol de trabajo 'v1'.

Te encuentras en estado 'detached HEAD'. Puedes revisar por aquí, hacer
cambios experimentales y confirmarlos, y puedes descartar cualquier
commit que hayas hecho en este estado sin impactar a tu rama realizando
otro checkout.

Si quieres crear una nueva rama para mantener los commits que has creado,
puedes hacerlo (ahora o después) usando -b con el comando checkout. Ejemplo:

  git checkout -b <nombre-de-nueva-rama>

HEAD está ahora en 8b670f6 Primer cambio en el repositorio
```

As we are informed in the message, right now we are working in *detached* mode (detached HEAD). This allows us to make the changes we want by creating commits without affecting the master branch.

It is advisable to run the `git checkout -b rama2` command now, because later we could forget about it, and when we change the branch we would lose the commits made.

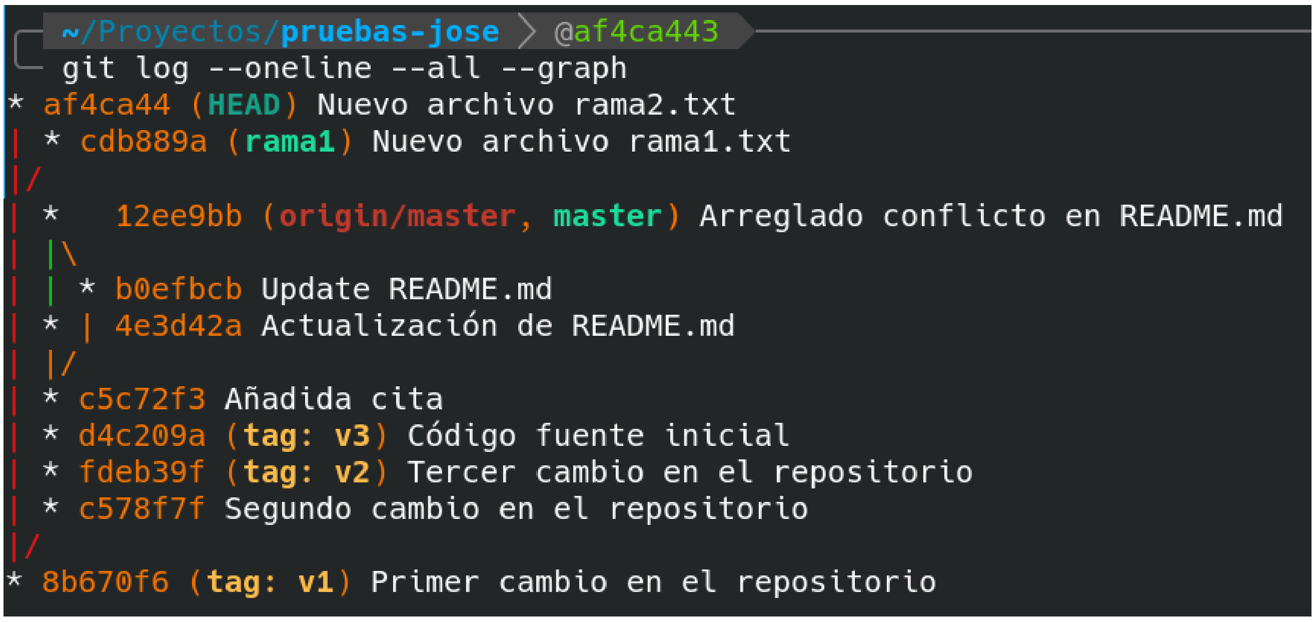
However, we are going to simulate that we forget to execute the previous command. We start making commits. In this case to simplify, we will only perform a commit.



```
~/Proyectos/pruebas-jose > #v1
nano rama2.txt

~/Proyectos/pruebas-jose > #v1 ?1
cat rama2.txt
Rama 2

~/Proyectos/pruebas-jose > #v1 ?1
git add rama2.txt

~/Proyectos/pruebas-jose > #v1 +1
git commit -m "Nuevo archivo rama2.txt"
[HEAD desacoplado af4ca44] Nuevo archivo rama2.txt
 1 file changed, 1 insertion(+)
 create mode 100644 rama2.txt
```

This creates a new commit for us. We run one more time

```
git log --oneline --all --graph
```

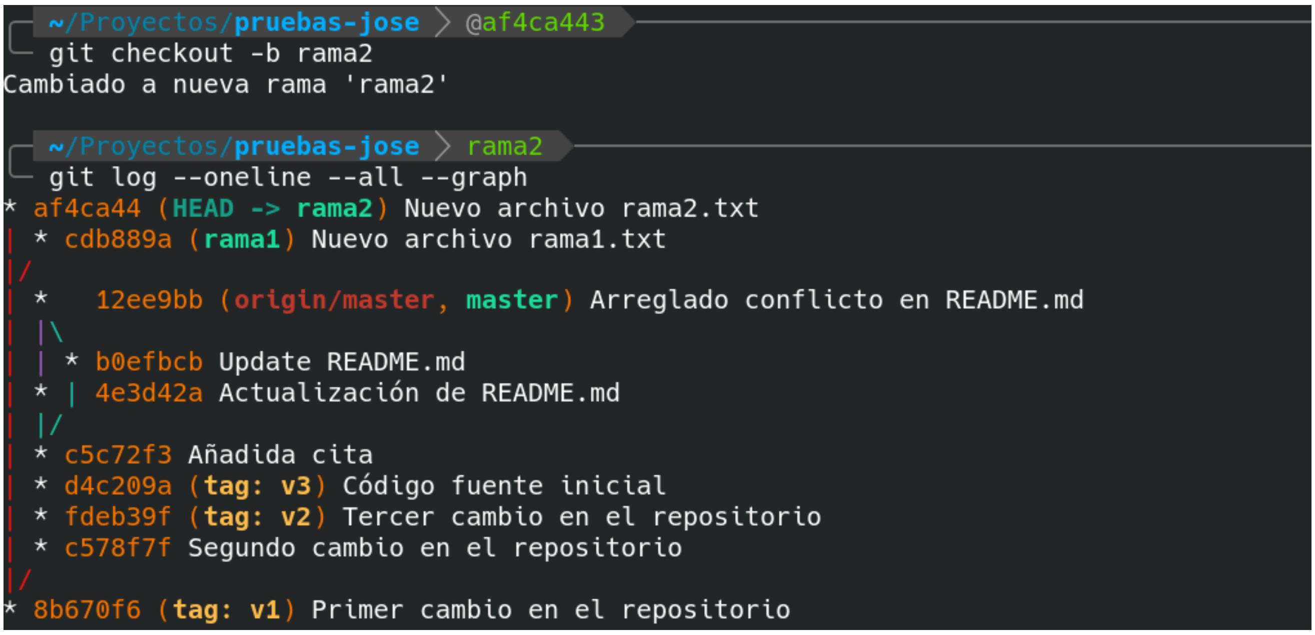


```
~/Proyectos/pruebas-jose > @af4ca443
git log --oneline --all --graph
* af4ca44 (HEAD) Nuevo archivo rama2.txt
| * cdb889a (rama1) Nuevo archivo rama1.txt
|/
| *   12ee9bb (origin/master, master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

As shown in the screenshot, there is no pointer in the form of a branch, so if we now, for example, execute `git checkout master`, we will lose all the commits made (in this case only one, but it could be many more).

If we do not want to lose these commits, we must execute

`git checkout -b rama2`



```
~/Proyectos/pruebas-jose > @af4ca443
git checkout -b rama2
Cambiado a nueva rama 'rama2'

~/Proyectos/pruebas-jose > rama2
git log --oneline --all --graph
* af4ca44 (HEAD -> rama2) Nuevo archivo rama2.txt
| * cdb889a (rama1) Nuevo archivo rama1.txt
|/
| *   12ee9bb (origin/master, master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

After this, we will be able to change branches with `git checkout` without fear of losing previous commits.

Make sure to run the above command before moving on to the next point.

10.2. Create branches with `git branch` ...

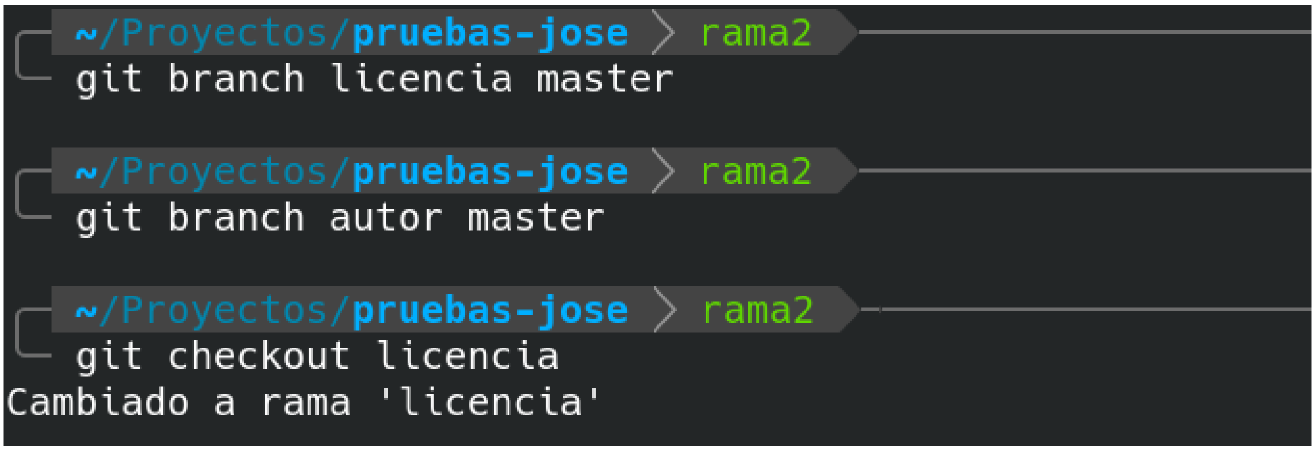
The command `git branch nueva-rama` It has essentially 2 forms:

1. `git branch new-branch` (We create a new branch from the current commit, but do NOT jump to it)
2. `git branch new-branch start-commit` (We create a new branch from the indicated commit, but we do NOT move to it)

After executing one of the above ways, we should always do a `git checkout` afterwards if we want to work with the new branch.

Let's see its use, making use of the second way. From the current branch, that is branch2, we are going to create 2 branches (called license and author) from the master branch.

```
git branch license master
git branch autor    master
```

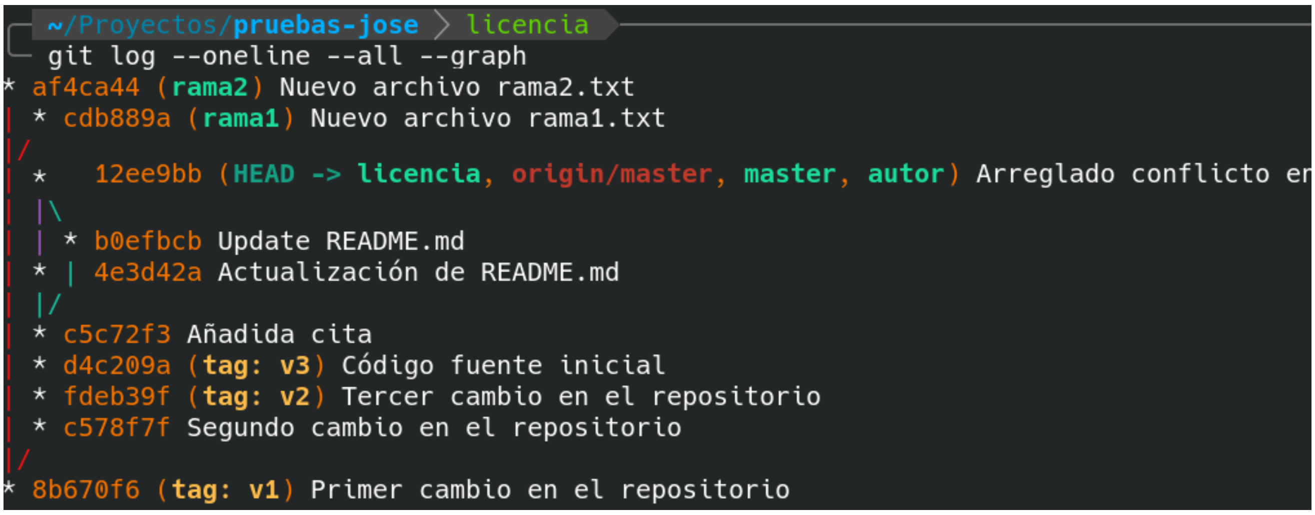


```
~/Proyectos/pruebas-jose > rama2
git branch licencia master

~/Proyectos/pruebas-jose > rama2
git branch autor master

~/Proyectos/pruebas-jose > rama2
git checkout licencia
Cambiado a rama 'licencia'
```

To start working with any of them, we must execute `git checkout ...` For example

```
git license checkout
```

With `git log --oneline --all --graph` we can see that the `HEAD` pointer now points to the `license` branch.



```
~/Proyectos/pruebas-jose 〉 licencia
 git log --oneline --all --graph
* af4ca44 (rama2) Nuevo archivo rama2.txt
| * cdb889a (rama1) Nuevo archivo rama1.txt
|/
| *   12ee9bb (HEAD -> licencia, origin/master, master, autor) Arreglado conflicto en
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

In this branch we will create a new file called `LICENSE`.

For this we do

```
nano LICENSE
```

We write inside a line with the following text: **GPL v3**

And we commit

```
git add LICENSE
git commit -m "New file LICENSE"
```

To work with the `author` branch, we run

```
git checkout autor
```

In this branch we are going to create an `AUTHOR` file and we are also going to modify the `README.md` file.

For this we do

```
nano AUTHOR
```

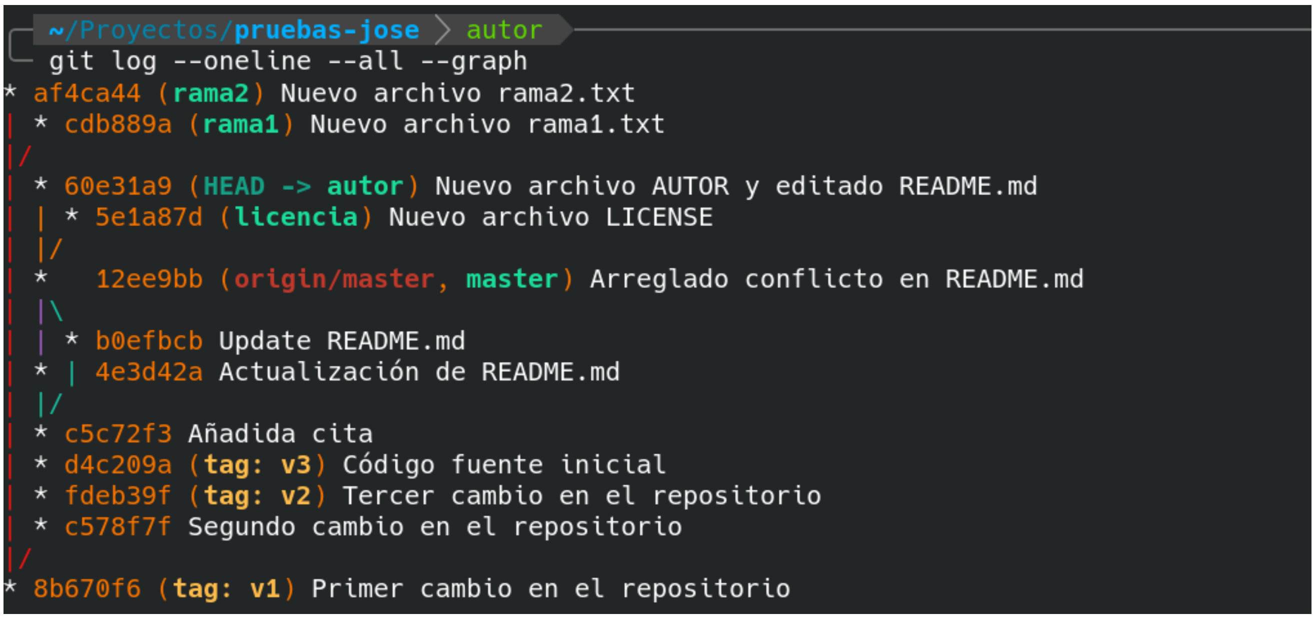
We write inside a line with the following text: **JOSÉ ANTONIO MUÑOZ JIMÉNEZ**

We will also modify the `README.md` file.
On the line where our name appears, we'll change the text to all caps.
The purpose is to cause a merge conflict in the future, which we will resolve in the next activity.

Y realizamos commit

```
git add AUTHOR
git commit -m "New AUTHOR file and edited README.md"
```

The result of `git log --oneline --all --graph` is

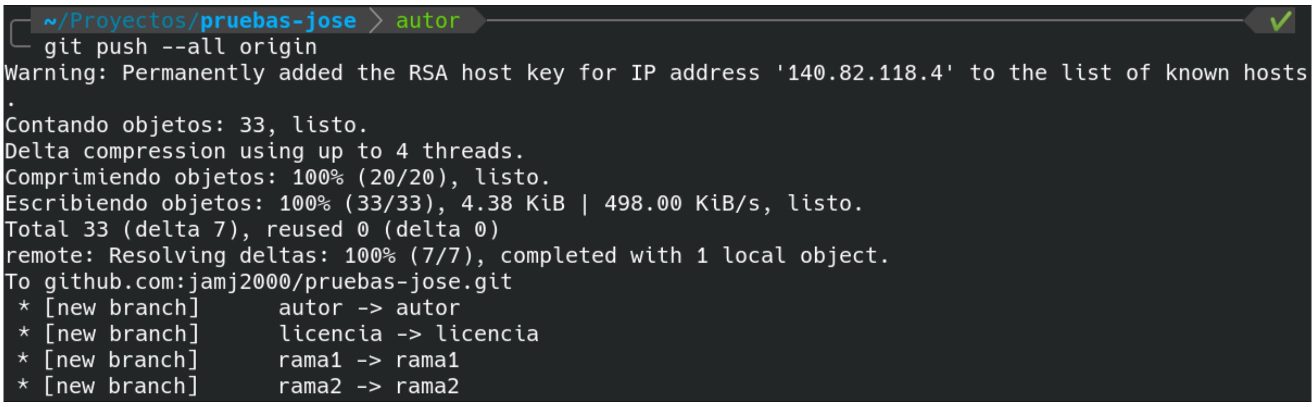


```
 ~/Proyectos/pruebas-jose > autor
└─ git log --oneline --all --graph
* af4ca44 (rama2) Nuevo archivo rama2.txt
| * cdb889a (rama1) Nuevo archivo rama1.txt
|/
| * 60e31a9 (HEAD -> autor) Nuevo archivo AUTOR y editado README.md
| | * 5e1a87d (licencia) Nuevo archivo LICENSE
| |/
| *   12ee9bb (origin/master, master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

10.3. Push branches to remote repository

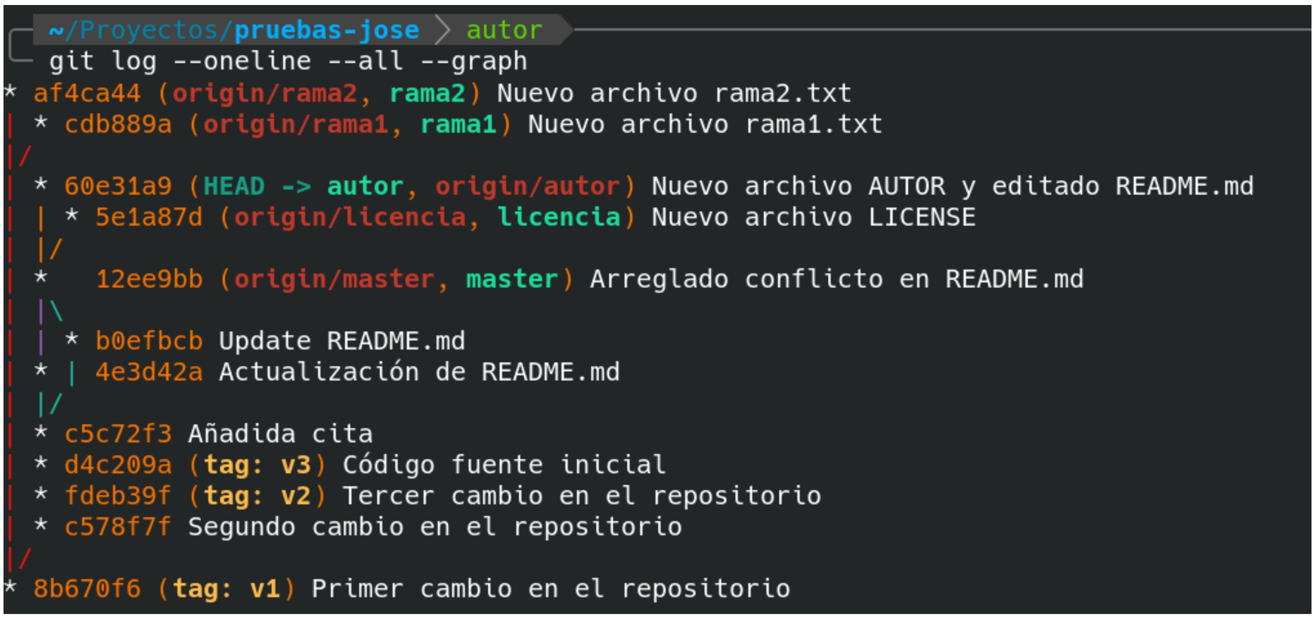
To upload all the changes made in all the branches we execute

`git push origin --all`



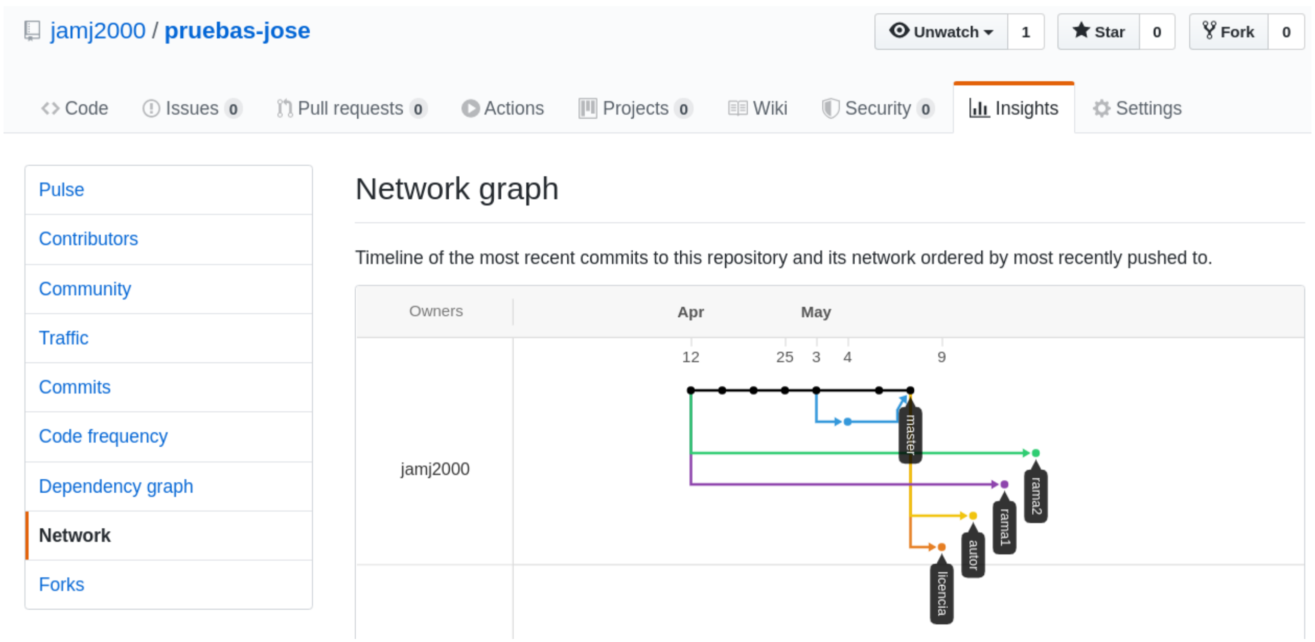
```
 ~/Proyectos/pruebas-jose > autor                                              ✓
└─ git push --all origin
Warning: Permanently added the RSA host key for IP address '140.82.118.4' to the list of known hosts
.
Contando objetos: 33, listo.
Delta compression using up to 4 threads.
Comprimiendo objetos: 100% (20/20), listo.
Escribiendo objetos: 100% (33/33), 4.38 KiB | 498.00 KiB/s, listo.
Total 33 (delta 7), reused 0 (delta 0)
remote: Resolving deltas: 100% (7/7), completed with 1 local object.
To github.com:jamj2000/pruebas-jose.git
 * [new branch]      autor -> autor
 * [new branch]      licencia -> licencia
 * [new branch]      rama1 -> rama1
 * [new branch]      rama2 -> rama2
```

The result is that all pointers to remote branches are updated (shown in red in the screenshot below)



```
~/Proyectos/pruebas-jose > autor
git log --oneline --all --graph
* af4ca44 (origin/rama2, rama2) Nuevo archivo rama2.txt
| * cdb889a (origin/rama1, rama1) Nuevo archivo rama1.txt
|/
| * 60e31a9 (HEAD -> autor, origin/autor) Nuevo archivo AUTOR y editado README.md
| | * 5e1a87d (origin/licencia, licencia) Nuevo archivo LICENSE
| |/
| *   12ee9bb (origin/master, master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

In **GitHub**, within the corresponding repository, we can see a graph of the branches by clicking on the `Insights` tab and then on the `Network` option (on the left side of the new page)



Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.

11. Merging and deleting branches

This activity is a continuation of the previous one. In it we will see how to perform **merge** branches and how to remove pointers to old branches.

Let's assume that we have worked on the branches of the previous activity `branch1`, `branch2`, `license` and `author`) by adding several more commits, although this has not really been the case. Branches with a single commit are usually not that frequent.

And it's time to discard the work done in some branch and integrate the content of others in the `master` branch.

In this activity we will discard the work done in `branch2`, and integrate the `branch1`, `license` and `author` branches into `master`.

To merge branches, use the command

`git merge ...`

11.1. Removing a local branch

To remove a local branch use the command

```
git branch -d  rama
```

For example, to delete `branch2` we do

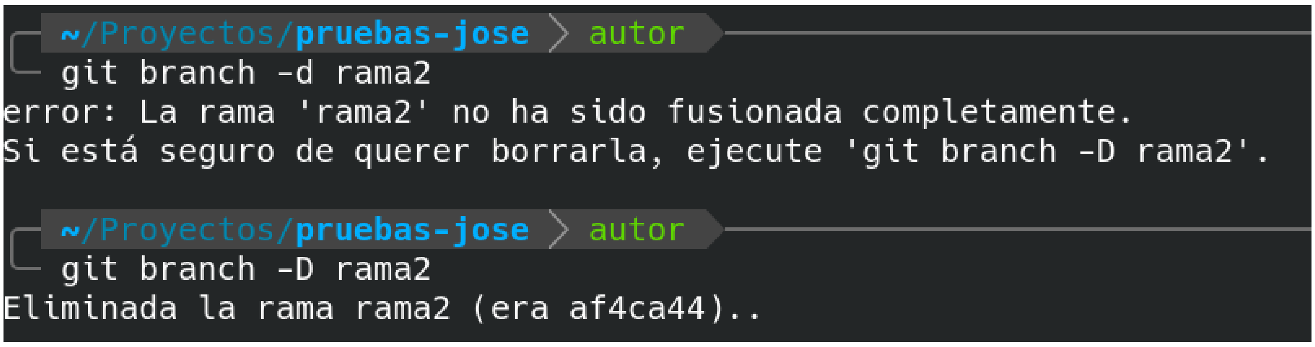
```
git branch -d rama2
```

The delete is not executed, since the changes have not been integrated into `master`, nor into any other branch.

To force the deletion we do

```
git branch -D rama2
```

In this way we lose all the modifications that we would have made in that branch.



```
~/Proyectos/pruebas-jose  autor
  git branch -d rama2
error: La rama 'rama2' no ha sido fusionada completamente.
Si está seguro de querer borrarla, ejecute 'git branch -D rama2'.

~/Proyectos/pruebas-jose  autor
  git branch -D rama2
Eliminada la rama rama2 (era af4ca44)..
```

11.2. Merging local branches

We are going to integrate the changes made in `branch1`, `license` and `author` into the `master` branch.

We will proceed as follows:

1. We switch to the `master` branch
2. We merge `licencia` branch
3. We merge `autor` branch
4. We merge branch `rama1`

11.2.1. Switch to `master` branch

It is **VERY IMPORTANT** to switch to the `master` branch. If we don't make the change, all merges would be done on the `author` branch (the branch we're currently on).

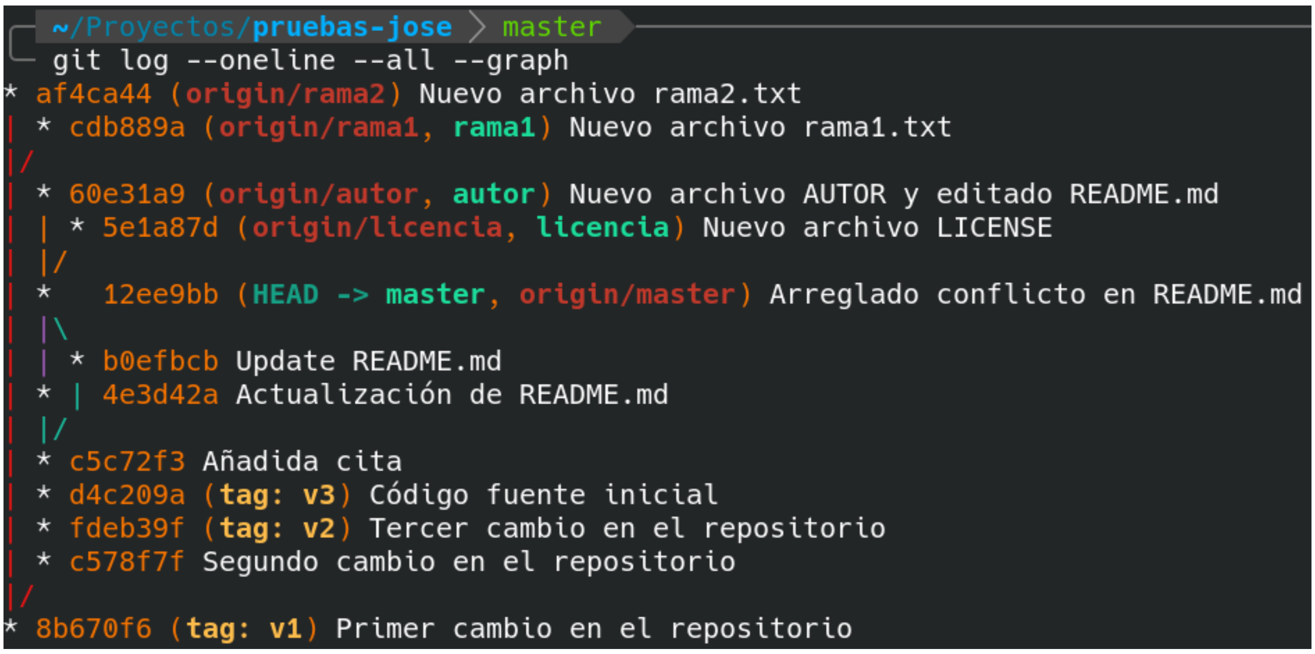
We must do

```
git  checkout master
```

11.2.2. We merge `licencia` branch

First, let's look at the structure of the branches. We make

```
git log --oneline --all --graph
```



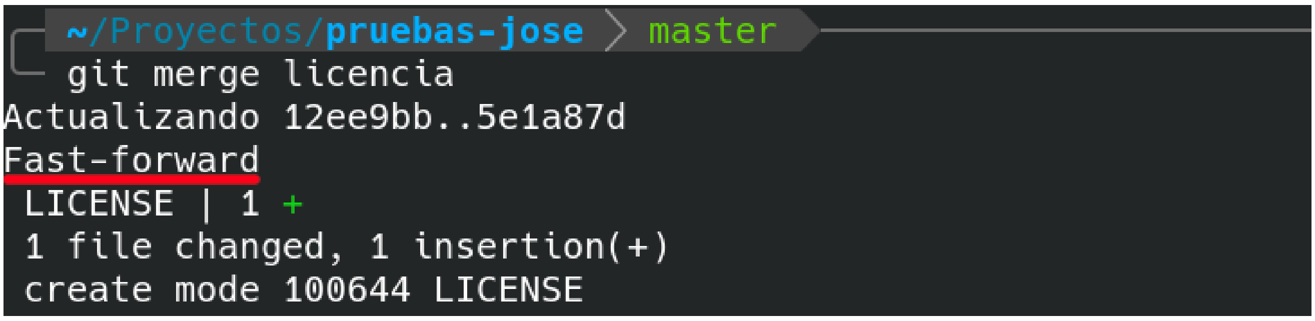
```
~/Proyectos/pruebas-jose > master
  git log --oneline --all --graph
* af4ca44 (origin/rama2) Nuevo archivo rama2.txt
| * cdb889a (origin/rama1, rama1) Nuevo archivo rama1.txt
|/
| * 60e31a9 (origin/autor, autor) Nuevo archivo AUTOR y editado README.md
| | * 5e1a87d (origin/licencia, licencia) Nuevo archivo LICENSE
| |/
| *   12ee9bb (HEAD -> master, origin/master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

Note that merging the `license` branch into the `master` branch is equivalent to moving the `HEAD` and `master` pointers up, ie matching them with the `license` pointer.

This type of merger is the simplest and never gives rise to conflicts. It is known as **fast-forward merge** (abbreviated **FF**) or **fast-forward merge**.

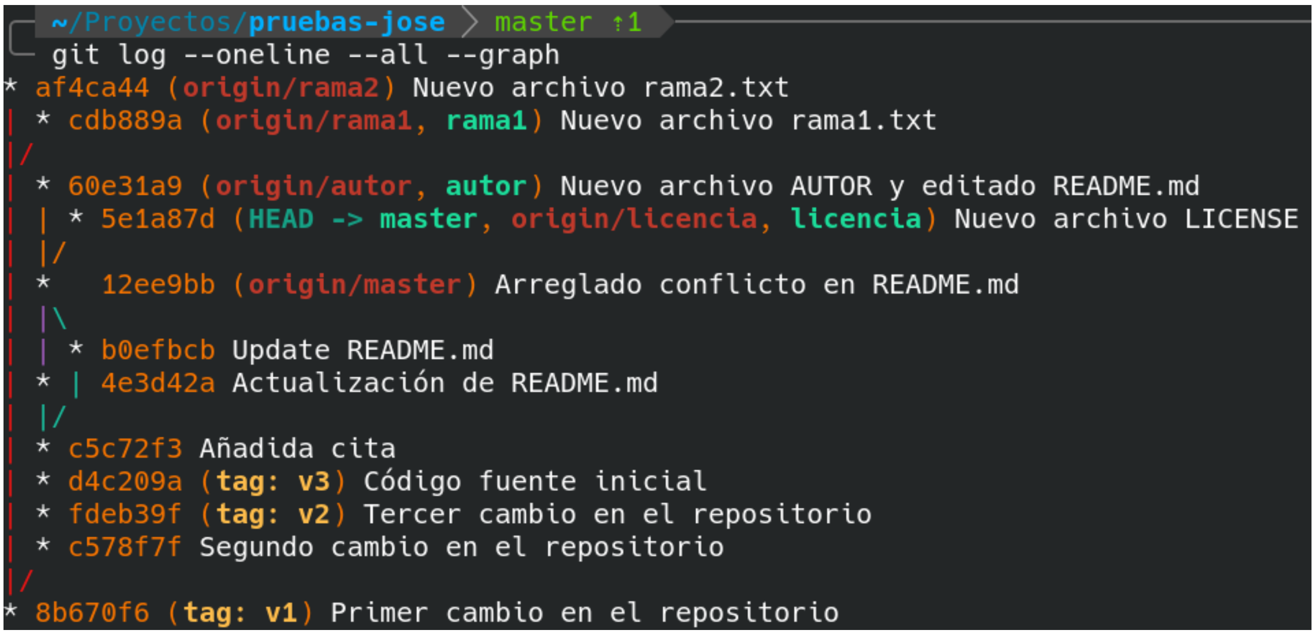
To merge this branch we do

```
git merge licencia
```



```
~/Proyectos/pruebas-jose > master
 git merge licencia
Actualizando 12ee9bb..5e1a87d
Fast-forward
 LICENSE | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 LICENSE
```

Observe how it is after the fusion. Only the `HEAD` and `master` pointers have been moved.



```
~/Proyectos/pruebas-jose > master ↑1
 git log --oneline --all --graph
* af4ca44 (origin/rama2) Nuevo archivo rama2.txt
| * cdb889a (origin/rama1, rama1) Nuevo archivo rama1.txt
|/
| * 60e31a9 (origin/autor, autor) Nuevo archivo AUTOR y editado README.md
| | * 5e1a87d (HEAD -> master, origin/licencia, licencia) Nuevo archivo LICENSE
| |/
| *   12ee9bb (origin/master) Arreglado conflicto en README.md
| |\
| | * b0efbcb Update README.md
| * | 4e3d42a Actualización de README.md
| |/
| * c5c72f3 Añadida cita
| * d4c209a (tag: v3) Código fuente inicial
| * fdeb39f (tag: v2) Tercer cambio en el repositorio
| * c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

NOTE: Don't worry right now about the remote pointers (the ones that appear in red). Later we will synchronize them with the remote repository.

11.2.3. Merge `autor` branch

If instead of merging a branch that is ahead of `master`, what we do is merge a branch that is in parallel with the `master` branch, then we will perform a **3-way merge**

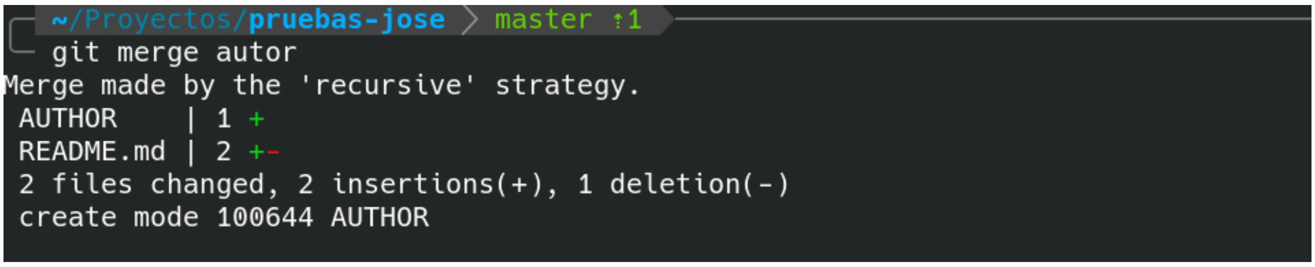
This type of merging can cause conflicts. If both branches contain modifications on the same lines in the same files a conflict may occur.

In this case, the `README.md` file has a line with the author's name, but with separate lines in the `master` and `author` branches (all caps).

To perform the merge we run

```
git merge autor
```

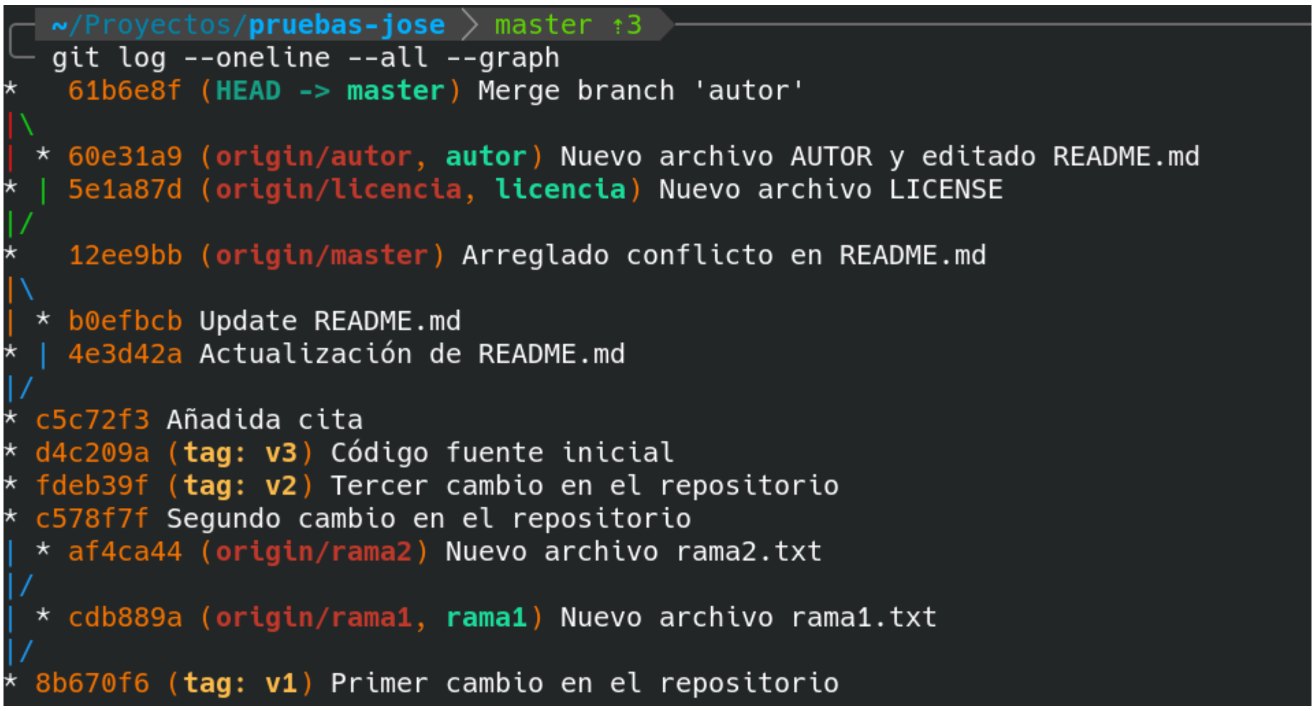
When the editor appears with the associated message, we will accept the message or edit it to our liking.



```
╭─ ~/Proyectos/pruebas-jose  master ↑1
╰─ git merge autor
Merge made by the 'recursive' strategy.
 AUTHOR    | 1 +
 README.md | 2 +-
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 AUTHOR
```

In this case, the conflict did not arise. It was automatically resolved in favor of the contents of the `author` branch.
Therefore the author in the `README.md` file will appear in all caps.

Notice how a new commit has been created as a result of merging the `author` branch and the `master` branch.
This always happens in the 3-way merge.



```
╭─ ~/Proyectos/pruebas-jose  master ↑3
╰─ git log --oneline --all --graph
*   61b6e8f (HEAD -> master) Merge branch 'autor'
|\
| * 60e31a9 (origin/autor, autor) Nuevo archivo AUTOR y editado README.md
* | 5e1a87d (origin/licencia, licencia) Nuevo archivo LICENSE
|/
*   12ee9bb (origin/master) Arreglado conflicto en README.md
|\
| * b0efbcb Update README.md
* | 4e3d42a Actualización de README.md
|/
* c5c72f3 Añadida cita
* d4c209a (tag: v3) Código fuente inicial
* fdeb39f (tag: v2) Tercer cambio en el repositorio
* c578f7f Segundo cambio en el repositorio
| * af4ca44 (origin/rama2) Nuevo archivo rama2.txt
|/
| * cdb889a (origin/rama1, rama1) Nuevo archivo rama1.txt
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

11.2.4. We merge branch branch1

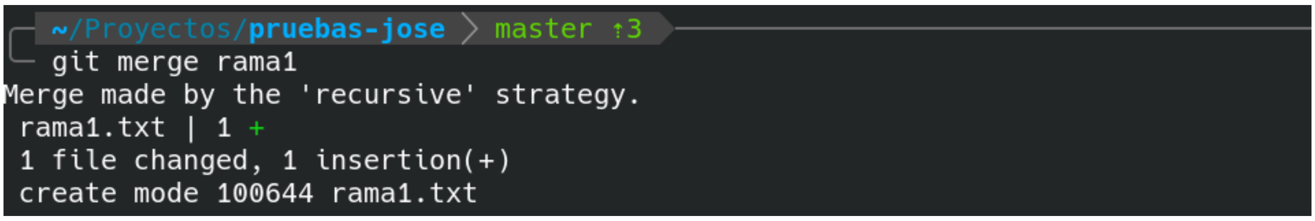
Finally, we will integrate the changes made in `branch1` into master.
It is a type of **3-way merger**, just like the previous one.

In this case, no conflict will occur, since in this branch we have only made changes
about the `branch1.txt` file, which does not exist in the `master` branch.

To perform the merger we do

```
git merge branch1
```

When the editor appears with the associated message, we will accept the message or edit it to our liking.

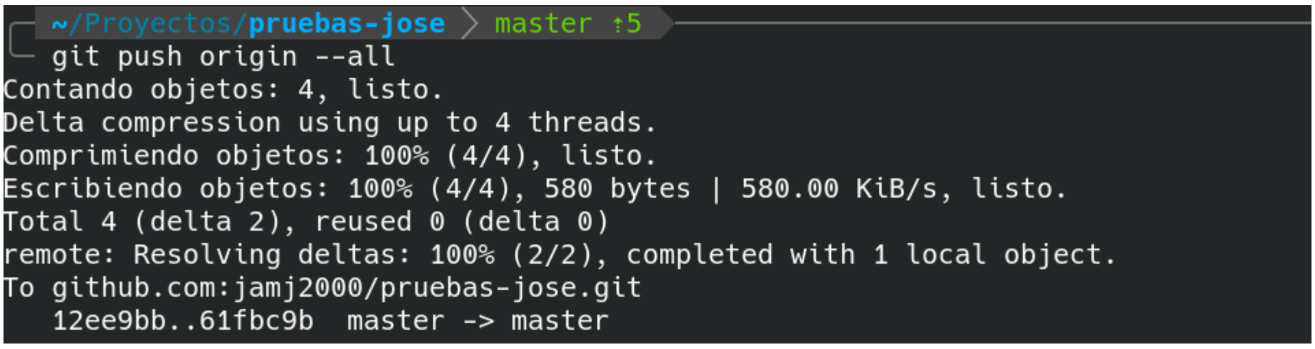


```
~/Proyectos/pruebas-jose  master ↑3
  git merge rama1
Merge made by the 'recursive' strategy.
 rama1.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 rama1.txt
```

11.3. Pushing changes to remote repository

To upload to the remote repository all the changes made in our local repository, we execute

```
git push origin --all
```



```
~/Proyectos/pruebas-jose  master ↑5
  git push origin --all
Contando objetos: 4, listo.
Delta compression using up to 4 threads.
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (4/4), 580 bytes | 580.00 KiB/s, listo.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:jamj2000/pruebas-jose.git
   12ee9bb..61fbc9b  master -> master
```

11.4. Removing pointers to local branches

To remove local pointers we run

```
git branch -d rama1
```

The pointers to `license` and `author` will not be removed, in case we wish to continue working on those branches in the future.

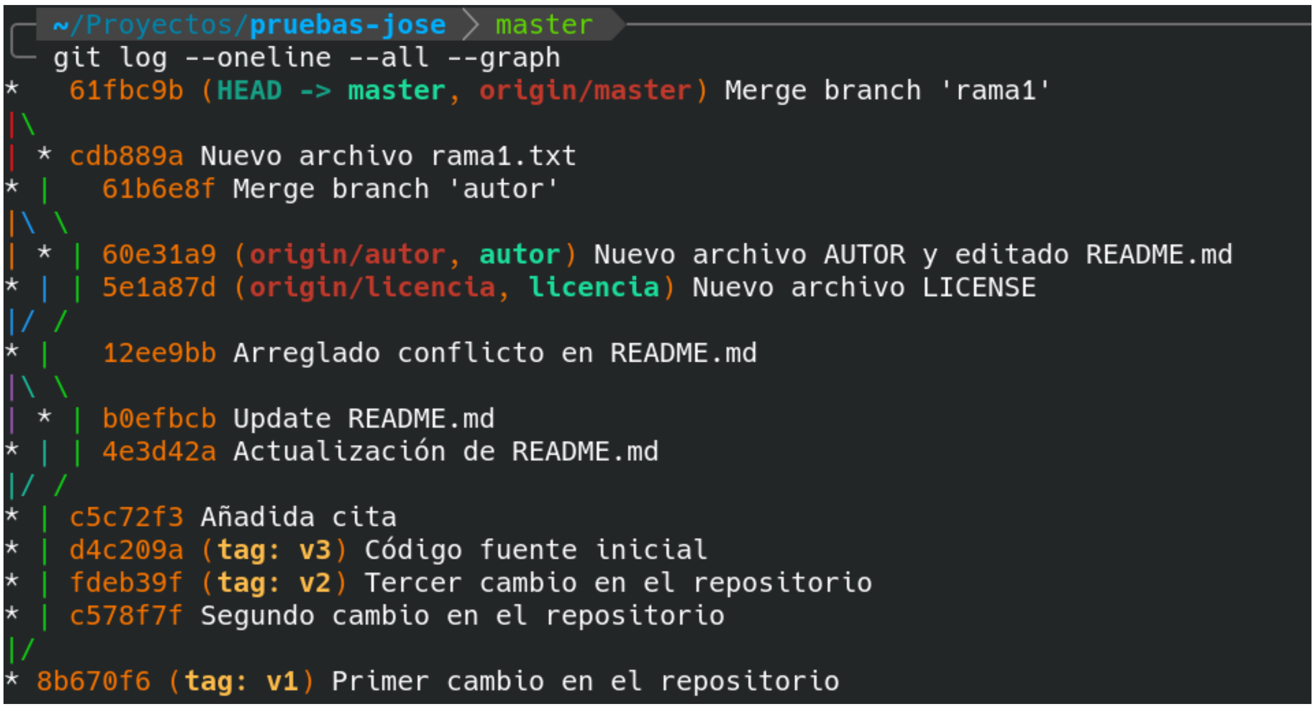
11.5. Removing pointers to remote branches

To remove the pointers in the remote repository, we run

```
git push origin --delete branch1
git push origin --delete branch2
```

The pointers to `origin/license` and `origin/author` will not be removed,
in case in the future we wish to continue working in these branches.

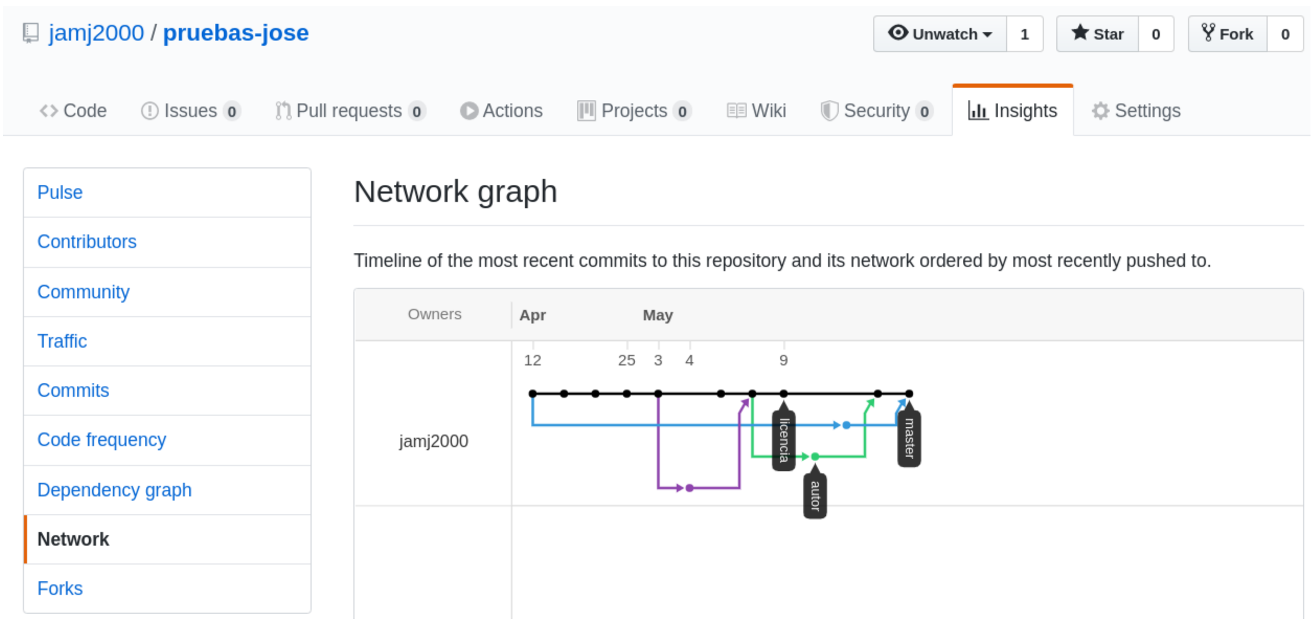
To see the status we run `git log ...`
See how the branches are up to date and in sync with the remote repository.



```
~/Proyectos/pruebas-jose 〉 master
git log --oneline --all --graph
*   61fbc9b (HEAD -> master, origin/master) Merge branch 'rama1'
|\
| * cdb889a Nuevo archivo rama1.txt
* |   61b6e8f Merge branch 'autor'
|\ \
| * | 60e31a9 (origin/autor, autor) Nuevo archivo AUTOR y editado README.md
* | | 5e1a87d (origin/licencia, licencia) Nuevo archivo LICENSE
|/ /
* |   12ee9bb Arreglado conflicto en README.md
|\ \
| * | b0efbcb Update README.md
* | | 4e3d42a Actualización de README.md
|/ /
* | c5c72f3 Añadida cita
* | d4c209a (tag: v3) Código fuente inicial
* | fdeb39f (tag: v2) Tercer cambio en el repositorio
* | c578f7f Segundo cambio en el repositorio
|/
* 8b670f6 (tag: v1) Primer cambio en el repositorio
```

11.6. Checking changes in remote repository

To see a graph of the branches in the remote repository, click on **Insights**, **Network**.



Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

11.7. Task proposed for the student

As a task, it is proposed

- go back to the `licencia` branch, add content to the `LICENSE` file and commit.
- go back to the `author` branch, add content to the `AUTHOR` file and commit.
- integrate changes from both branches into the `master` branch.

NOTE: Do not delete the local repository or remote repository. We will use them again in the next activity.

Upload to the AULES platform a PDF document with the relevant screenshots and explanations.