



DOCUMENTACIÓN GIT

Autor: Daniel Fernández Balsera

ORGANIZACIÓN CODIGO FUENTE Y LOS ELEMENTOS DEL PROYECTO

CREACION REPOSITORIO

1. Creamos la carpeta donde vamos a crear nuestro repositorio

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git  
$ mkdir escenario1  
  
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git  
$ cd escenario1/
```

2. Inicializamos el repositorio

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1  
$ git init  
Initialized empty Git repository in C:/Users/danif/Desktop/curso_git/escenario1/.git/
```

3. Creación del fichero readme.md

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)  
$ echo "# Primera linea de un fichero readme.md" > readme.md
```

4. Si hacemos un git status, comprobaremos que aún no se ha realizado ninguna revisión, no hemos ningún commit, etc.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.md

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

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```

5. Si realizamos un git add readme.md, al hacer el git status podemos comprobar que ya tenemos el fichero readme.md en el área de preparación.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git add readme.md
warning: in the working copy of 'readme.md', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```

6. Creamos nuestro primer commit.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git commit -m "Este es el mensaje del primer commit de dani fernandez"
[master (root-commit) 8fb1172] Este es el mensaje del primer commit de dani fernandez
1 file changed, 1 insertion(+)
create mode 100644 readme.md
```

7. Para poder realizar lo anterior, es importante que escribáis las dos siguientes líneas de comandos.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git config --global user.email "daniseneca.df@gmail.com"

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git config --global user.name "DaniBalsera"
```

8. Si ahora hacemos git status dice que estamos en la rama master y que el directorio de trabajo está limpio, es decir, no hay cambios pendientes de aprobar.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git status
On branch master
nothing to commit, working tree clean

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ !
```

9. Si hacemos un git log --oneline, podemos ver que ya tenemos una revisión.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git log --oneline
8fb1172 (HEAD -> master) Este es el mensaje del primer commit de dani fernandez
```

8fb1172 es un identificador de versiones.

PROBANDO CAMBIOS EN EL REPOSITORIO

1. Vamos a crear varios ficheros tmp, es muy común que tengamos muchos ficheros, y usemos un comodín para incluir todos los directorios que tengamos en nuestro directorio de trabajo.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ echo "Fichero 1" >> fichero1.tmp

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ echo "Fichero 2" >> fichero2.tmp

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ echo "Fichero 3" >> fichero3.tmp

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ echo "Fichero 4" >> fichero4.tmp

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```

Aspecto actual del directorio:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ ls -la
total 17
drwxr-xr-x 1 danif 197609  0 Mar 18 12:08 ./
drwxr-xr-x 1 danif 197609  0 Mar 18 10:19 ../
drwxr-xr-x 1 danif 197609  0 Mar 18 11:59 .git/
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero1.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero2.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero3.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:08 fichero4.tmp
-rw-r--r-- 1 danif 197609 40 Mar 18 10:22 readme.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```

2. Si hacemos un git status, los ficheros aparecerán en rojo porque están sin revisar, pero tenemos la opción de crear una máscara dentro de un fichero llamado .gitignore, en este fichero especificaremos que todos los ficheros que sean *.tmp, no se metan en el area de staging (área donde se encuentran los datos del proyecto).

```
GNU nano 7.1 .gitignore
*.tmp|
```

Esto suele usarse para ficheros que no vayamos a querer en una revisión.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ ls -la
total 18
drwxr-xr-x 1 danif 197609  0 Mar 18 12:14 ./
drwxr-xr-x 1 danif 197609  0 Mar 18 10:19 ../
drwxr-xr-x 1 danif 197609  0 Mar 18 11:59 .git/
-rw-r--r-- 1 danif 197609  6 Mar 18 12:14 .gitignore
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero1.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero2.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero3.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:08 fichero4.tmp
-rw-r--r-- 1 danif 197609 40 Mar 18 10:22 readme.md
```

3. Si escribimos **git add .** sería decirle a git que incluya todos lo ficheros que tenga en el working directory, en el área de staging.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git add .
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
```

Si hacemos un git status, el único fichero que no tenemos en revisión es el .gitignore.

4. Si queremos crear una nueva revisión para el .gitignore, hay que realizar lo siguiente:

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git add .gitignore

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git commit -m "Nueva revisión para el fichero: .gitignore"
[master db42561] Nueva revisión para el fichero: .gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git status
On branch master
nothing to commit, working tree clean

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ !
```

5. Comprobamos la revisión del último commit.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git log --oneline
db42561 (HEAD -> master) Nueva revisión para el fichero: .gitignore
8fb1172 Este es el mensaje del primer commit de dani fernandez
```

Al hacer un `ls -l`, .gitignore no aparece ya que en linux este tipo de ficheros son ocultos.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ ls -l
total 5
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero1.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero2.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:07 fichero3.tmp
-rw-r--r-- 1 danif 197609 10 Mar 18 12:08 fichero4.tmp
-rw-r--r-- 1 danif 197609 40 Mar 18 10:22 readme.md
```

- Ahora vamos a hacer algunas pruebas generales, en la siguientes líneas de comandos, vais a ver, como creo un nuevo fichero y lo agrego al área de staging, como modifico el readme.md y me avisa para que lo actualice, y finalmente, haré un git add. para agregar actualizaciones y inicializaré un commit, finalmente mostraré las 3 revisiones que hemos realizado hasta ahora.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ echo "Este es mi segundo fichero readme.md" >> readme2.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ nano readme.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme2.md

no changes added to commit (use "git add" and/or "git commit -a")

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git add .
warning: in the working copy of 'readme.md', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'readme2.md', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git commit -m "Esta es la tercera revisión que estamos realizando"
[master d61fba0] Esta es la tercera revisión que estamos realizando
 2 files changed, 2 insertions(+)
 create mode 100644 readme2.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git log --oneline
d61fba0 (HEAD -> master) Esta es la tercera revisión que estamos realizando
db42561 Nueva revisión para el fichero: .gitignore
8fb1172 Este es el mensaje del primer commit de dani fernandez

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```

7. Ahora usaremos la herramienta diff, para que nos explique cuál es la diferencia entre los ficheros que se están modificando dentro de nuestro directorio.

Por ejemplo, vamos a modificar el fichero readme.md:

Contenido original:

```
GNU nano 7.1                               readme.md
# Primera línea de un fichero readme.md
# Segunda línea de un fichero readme.md
```

Contenido actual:

```
GNU nano 7.1                               readme.md
# Primera línea de un fichero readme.md
# Segunda línea de un fichero readme.md para DaniBalsera|
```

Ahora hacemos uso de la herramienta diff.

Esta herramienta nos muestra en rojo la línea original que se ha modificado, y en verde el resultado de la modificación.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ git diff
warning: in the working copy of 'readme.md', LF will be replaced by CRLF the next time Git touches it
diff --git a/readme.md b/readme.md
index d332dc5..1deba75 100644
--- a/readme.md
+++ b/readme.md
@@ -1,2 +1,2 @@
 # Primera línea de un fichero readme.md
-# Segunda línea de un fichero readme.md
+# Segunda línea de un fichero readme.md para DaniBalsera

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario1 (master)
$ |
```


TRABAJO EN REMOTO CON GITHUB

Para este apartado vamos a crear un escenario nuevo y vamos a clonar algún repositorio existente en github.

(Como en los apartados anteriores expliqué la inicialización del repositorio, no he visto necesario poner esos pasos).

Vais a vincularos con el siguiente proyecto: <https://github.com/sharkdp/bat>

1. Vinculación con el proyecto de github.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario2 (master)
$ git remote add origin https://github.com/sharkdp/bat

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario2 (master)
$
```

2. Vamos recuperar las diferentes ramas,archivos,datos,etc de dicho proyecto.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario2 (master)
$ git fetch origin
remote: Enumerating objects: 14370, done.
remote: Counting objects: 100% (115/115), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 14370 (delta 50), reused 79 (delta 31), pack-reused 14255
Receiving objects: 100% (14370/14370), 29.36 MiB | 11.85 MiB/s, done.
Resolving deltas: 100% (9579/9579), done.
From https://github.com/sharkdp/bat
 * [new branch]      bat-plugins      -> origin/bat-plugins
 * [new branch]      ci-experiment    -> origin/ci-experiment
 * [new branch]      create_highlighted_versions-wrong-path-repro -> origin/create_highlighted_versions-wrong-path-repro
 * [new branch]      crontab          -> origin/crontab
 * [new branch]      master           -> origin/master
 * [new branch]      release-v0.18.3 -> origin/release-v0.18.3
 * [new branch]      sponsor          -> origin/sponsor
 * [new tag]         v0.18.3          -> v0.18.3
 * [new tag]         v0.1.0          -> v0.1.0
 * [new tag]         v0.10.0         -> v0.10.0
 * [new tag]         v0.11.0         -> v0.11.0
 * [new tag]         v0.12.0         -> v0.12.0
 * [new tag]         v0.12.1         -> v0.12.1
 * [new tag]         v0.13.0         -> v0.13.0
 * [new tag]         v0.14.0         -> v0.14.0
 * [new tag]         v0.15.0         -> v0.15.0
 * [new tag]         v0.15.1         -> v0.15.1
 * [new tag]         v0.15.2         -> v0.15.2
 * [new tag]         v0.15.3         -> v0.15.3
 * [new tag]         v0.15.4         -> v0.15.4
 * [new tag]         v0.16.0         -> v0.16.0
 * [new tag]         v0.17.0         -> v0.17.0
 * [new tag]         v0.17.1         -> v0.17.1
 * [new tag]         v0.18.0         -> v0.18.0
 * [new tag]         v0.18.1         -> v0.18.1
 * [new tag]         v0.18.2         -> v0.18.2
 * [new tag]         v0.19.0         -> v0.19.0
 * [new tag]         v0.2.0          -> v0.2.0
 * [new tag]         v0.2.1          -> v0.2.1
 * [new tag]         v0.2.2          -> v0.2.2
 * [new tag]         v0.2.3          -> v0.2.3
 * [new tag]         v0.20.0         -> v0.20.0
 * [new tag]         v0.21.0         -> v0.21.0
 * [new tag]         v0.22.0         -> v0.22.0
 * [new tag]         v0.22.1         -> v0.22.1
 * [new tag]         v0.3.0          -> v0.3.0
```

IMPORTANTE: Sólo hemos recuperado los archivos, pero aún no los hemos descargado a nuestra área de staging.

3. Vamos a trabajar ahora con el comando **git pull** para descargar desde origin y que nos lo descargue en nuestra rama master.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario2 (master)
$ git pull origin master
From https://github.com/sharkdp/bat
* branch      master      -> FETCH_HEAD
Updating files: 100% (833/833), done.
```

Si ahora realizamos un `ls -l`, vamos a observar que se han descargado los archivos del proyecto.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario2 (master)
$ ls -l
total 166
-rw-r--r-- 1 danif 197609 43060 Mar 18 12:58 CHANGELOG.md
-rw-r--r-- 1 danif 197609 2541 Mar 18 12:58 CONTRIBUTING.md
-rw-r--r-- 1 danif 197609 39875 Mar 18 12:58 Cargo.lock
-rw-r--r-- 1 danif 197609 2957 Mar 18 12:58 Cargo.toml
-rw-r--r-- 1 danif 197609 11558 Mar 18 12:58 LICENSE-APACHE
-rw-r--r-- 1 danif 197609 1116 Mar 18 12:58 LICENSE-MIT
-rw-r--r-- 1 danif 197609 254 Mar 18 12:58 NOTICE
-rw-r--r-- 1 danif 197609 28907 Mar 18 12:58 README.md
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 assets/
-rw-r--r-- 1 danif 197609 3498 Mar 18 12:58 build.rs
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 diagnostics/
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 doc/
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 examples/
-rw-r--r-- 1 danif 197609 21 Mar 18 12:58 rustfmt.toml
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 src/
drwxr-xr-x 1 danif 197609 0 Mar 18 12:58 tests/

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario2 (master)
```

Esto sería un ejemplo de como traer a mi repositorio local, datos de un proyecto en github de la siguiente forma:

1. Git init
2. Git push
3. Git fetch

En el siguiente escenario que vamos a crear, vamos a trabajar con **git clone** , que es una forma más rápida de realizar una conexión remota con un proyecto de github.

Crearéis un escenario3 y lo inicializaréis.

1. Yo en este caso voy a clonar un repositorio que tengo creado en github, se puede hacer también con gitlab.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git clone https://github.com/DaniBalseira/ApiDaniFernandez
Cloning into 'ApiDaniFernandez'...
remote: Enumerating objects: 91, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (86/86), done.
remote: Total 91 (delta 23), reused 54 (delta 2), pack-reused 0
Receiving objects: 100% (91/91), 309.68 KiB | 1.31 MiB/s, done.
Resolving deltas: 100% (23/23), done.
```

2. Como podemos observar, se ha clonado todo perfectamente.

Ahora el siguiente paso es que en vez de estar en la carpeta **ApiDaniFernandez** , se mueva a la carpeta **escenario3**.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ ls -lart
total 12
drwxr-xr-x 1 danif 197609 0 Mar 18 13:08 ../
drwxr-xr-x 1 danif 197609 0 Mar 18 13:08 .git/
drwxr-xr-x 1 danif 197609 0 Mar 18 13:14 ./
drwxr-xr-x 1 danif 197609 0 Mar 18 13:14 ApiDaniFernandez/

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ cd ApiDaniFernandez/

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3/ApiDaniFernandez (master)
$ ls
README.md      karma.conf.js  src/           tsconfig.json
angular.json   package-lock.json  tsconfig.app.json  tsconfig.spec.json
e2e/           package.json    tsconfig.base.json  tslint.json

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3/ApiDaniFernandez (master)
$ |
```

Tendríamos que borrar el proyecto **ApiDaniFernandez**, y tendríamos que volver a clonarlo pero de manera específica.

3. Borrado de un proyecto clonado y clonación en una carpeta específica.

Para borrar un proyecto clonado de github, habría que escribir la siguiente línea de comandos.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ rm -rf ApiDaniFernandez/
```

Ahora vamos a clonar de nuevo el proyecto pero de manera especificada.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3
$ git clone https://github.com/DaniBalsera/ApiDaniFernandez .
Cloning into '.'...
remote: Enumerating objects: 91, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (86/86), done.
remote: Total 91 (delta 23), reused 54 (delta 2), pack-reused 0
Receiving objects: 100% (91/91), 309.68 KiB | 2.06 MiB/s, done.
Resolving deltas: 100% (23/23), done.
```

Aviso: Posiblemente os pueda salir el siguiente error cuando ejecutáis el anterior comando:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git clone https://github.com/DaniBalsera/ApiDaniFernandez .
fatal: destination path '.' already exists and is not an empty directory.
```

Solución:

rm -rf .*

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git clone https://github.com/DaniBalsera/ApiDaniFernandez .
fatal: destination path '.' already exists and is not an empty directory.

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ rm -rf .*
rm: unknown option -- .
Try 'rm --help' for more information.

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ rm -rf .*

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3
$ git clone https://github.com/DaniBalsera/ApiDaniFernandez .
Cloning into '.'...
remote: Enumerating objects: 91, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (86/86), done.
remote: Total 91 (delta 23), reused 54 (delta 2), pack-reused 0
Receiving objects: 100% (91/91), 309.68 KiB | 2.06 MiB/s, done.
Resolving deltas: 100% (23/23), done.
```

Si ponéis la anterior línea, os vaciara todo y ya podréis clonar en vuestro repositorio.

4. Vamos a crear y añadir un fichero nuevo llamado README3 y este lo subiremos al repositorio de github.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ nano README3

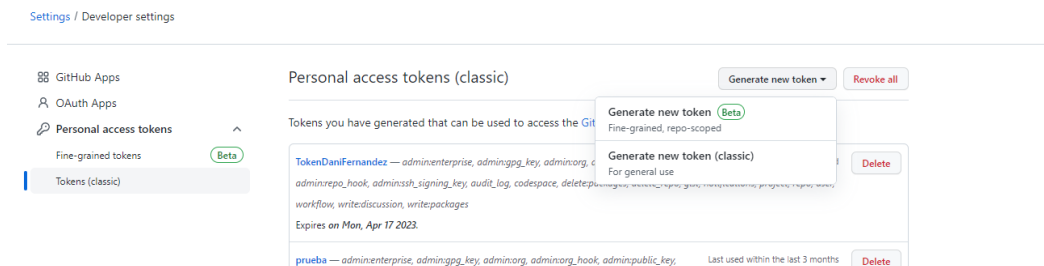
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git add README3
warning: in the working copy of 'README3', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git commit -m "Revision rellenando el nuevo readme3"
[master 2338057] Revision rellenando el nuevo readme3
1 file changed, 1 insertion(+)
create mode 100644 README3

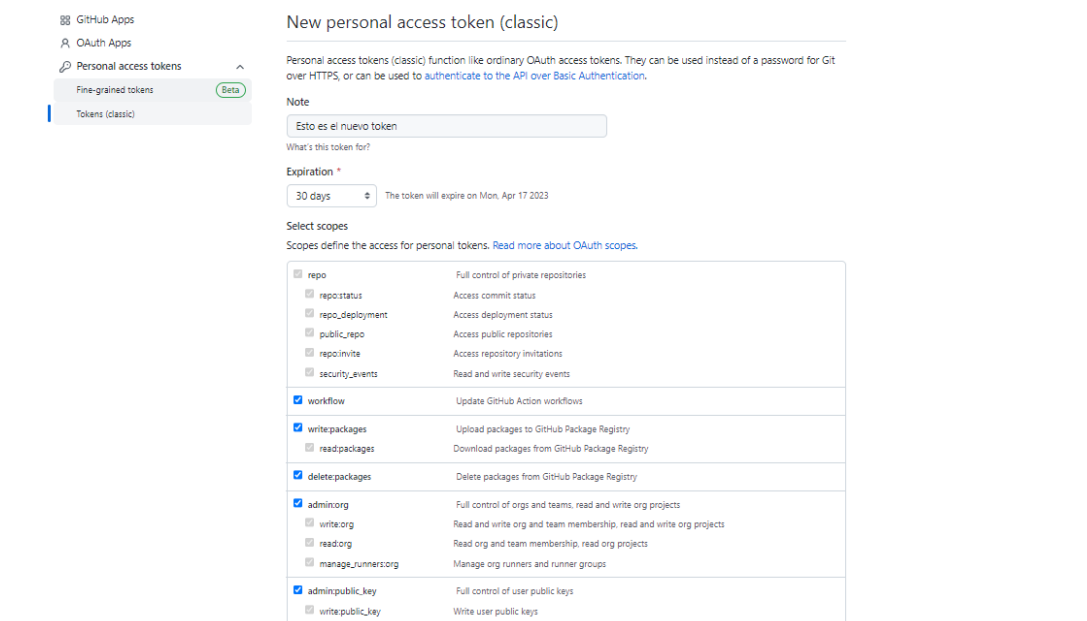
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ |
```

Antes de subirlo tenemos que obtener las credenciales para poder subir la información al repositorio de github.

Para generar un nuevo token, tenemos que ir a Settings/Developer settings y le damos a generarlo de forma clásica.



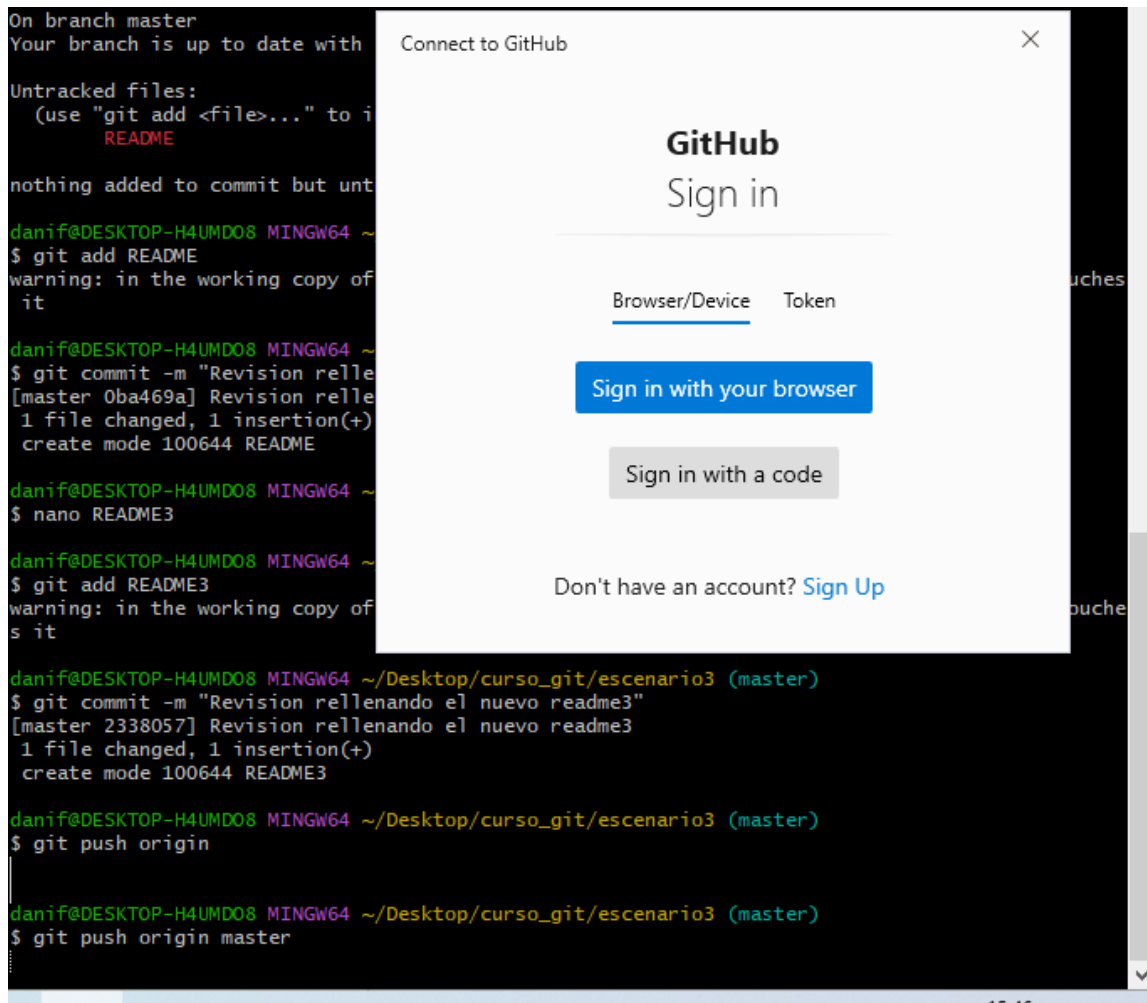
Aceptamos todas las opciones y le ponemos una nota:



Una vez hecho eso, copiamos la clave, y realizamos el siguiente paso:

Ponemos: **git push origin master**

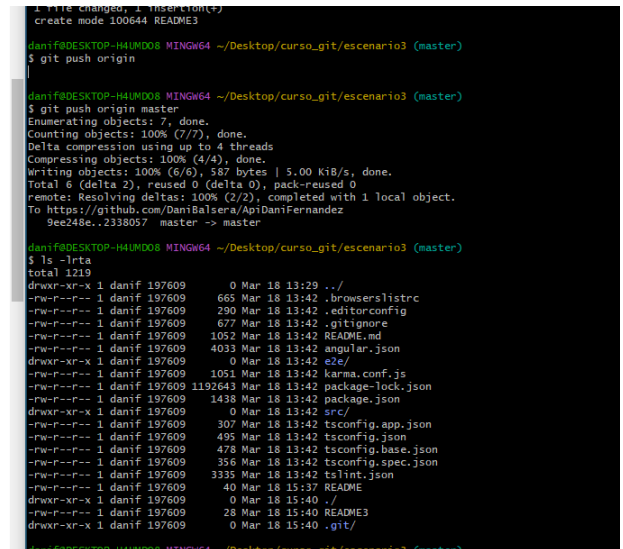
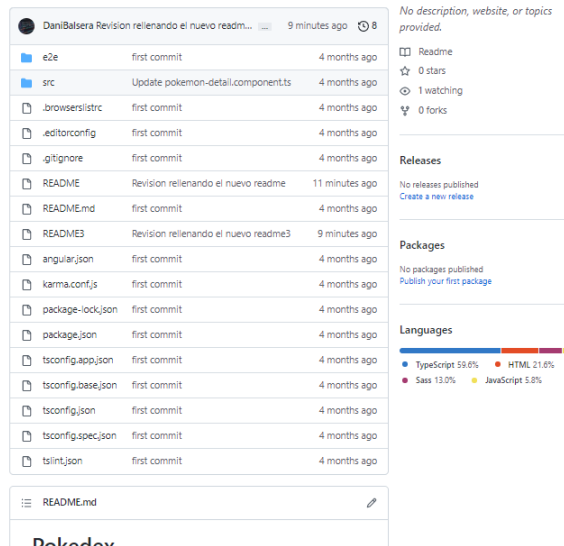
Podemos iniciar sesión tanto por navegador como con un código.



```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario3 (master)
$ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 587 bytes | 5.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/DaniBalsera/ApiDaniFernandez
9ee248e..2338057 master -> master
```

Ahora vamos a comprobar que el contenido de README3 se ha subido a mi proyecto.

Efectivamente tenemos el mismo contenido.

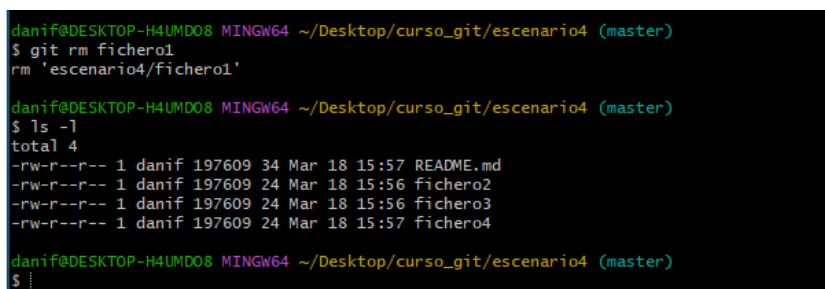


DESHACIENDO CAMBIOS CON GIT

Para esto vamos a trabajar con un nuevo escenario, escenario4.

Vamos a simular que hacemos algo que no queremos a hacer y vamos a recuperarnos de ese problema.

1. Por ejemplo vamos a recuperar un fichero que hemos borrado pero se ha revisado.



Vamos a recuperar ese archivo:

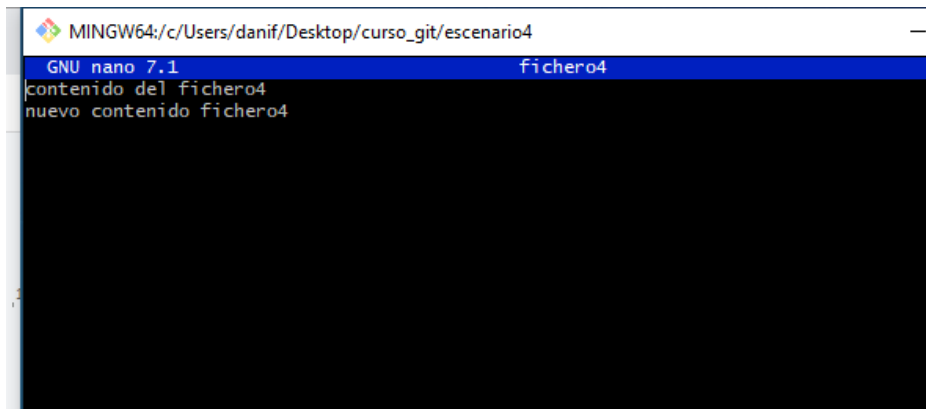
HEAD -> Etiqueta de la revisión (`git log --oneline`)

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git checkout HEAD -- fichero1

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ ls
README.md fichero1 fichero2 fichero3 fichero4
```

2. Ahora vamos a hacer modificaciones en un fichero.

Imaginad que cogemos el fichero 4 y le agregamos una nueva línea:



```
MINGW64:/c:/Users/danif/Desktop/curso_git/escenario4
GNU nano 7.1 fichero4
contenido del fichero4
nuevo contenido fichero4
```

Vamos a realizar un `git status`.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fichero4
```

Ahora realizaremos un `git add`.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git add .
warning: in the working copy of 'escenario4/fichero4', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   fichero4
```


Pero ahora resulta que no queríamos haber realizado el git add del fichero4, si no del 3, pero ahora los cambios están actualizados, pues muy sencillo, habría que realizar lo siguiente:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git reset HEAD fichero4
Unstaged changes after reset:
M      escenario4/fichero4

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fichero4
```

Todo vuelve a su estado original en lo que respecta al fichero4.

3. Ahora vamos a hacer modificaciones en varios ficheros a la vez, por ejemplo todos los ficheros tiene añadidos una ultima línea en la que pone lo mismo.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ echo "ultima linea equivalente" >> fichero1

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ echo "ultima linea equivalente" >> fichero2

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ echo "ultima linea equivalente" >> fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ echo "ultima linea equivalente" >> fichero4

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fichero1
        modified:   fichero2
        modified:   fichero3
        modified:   fichero4
```

Voy a realizar un git add .

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git add .
warning: in the working copy of 'escenario4/fichero1', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'escenario4/fichero2', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'escenario4/fichero3', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'escenario4/fichero4', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ |
```

Ahora escribimos la siguiente línea de código para deshacer los cambios:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git reset --hard HEAD~1
HEAD is now at f722a5a commit del escenario 4
```

Ahora imagina que hacemos lo mismo, volvemos a introducir una última línea a cada fichero pero queremos deshacer los cambios de manera más rápida, pues sería tan sencillo como hacer un revert al commit.

El comando sería: git revert (id del commit, en este caso: fddf71a).

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git commit -m "he incluido una ultima linea en todos los ficheros"
[master fddf71a] he incluido una ultima linea en todos los ficheros
4 files changed, 4 insertions(+)

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ git log --oneline
fddf71a (HEAD -> master) he incluido una ultima linea en todos los ficheros
f722a5a commit del escenario 4

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
[master 908f9fc] Revert "he incluido una ultima linea en todos los ficheros"
4 files changed, 4 deletions(-)

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ cat fichero1
contenido del fichero1

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ cat fichero2
contenido del fichero2

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ cat fichero3
contenido del fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$ cat fichero4
contenido del fichero4

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario4 (master)
$
```

FLUJOS DE TRABAJO

EXPERIMENTANDO CON RAMAS

Para esto vamos a crear un nuevo escenario, escenario5, además de vincular dicho repositorio con un proyecto que tengamos creado en github, esto es algo que hemos realizado en apartados anteriores, así que os vendrá bien para ponerlo en práctica.

1. Vamos a crear una rama.

new_master: La rama que creamos.

master: La rama de la que partimos.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git branch new_master master
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
```

2. Ahora vamos a movernos a la nueva rama.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git checkout new_master
Switched to branch 'new_master'
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ |
```

3. Vamos a agregar nueva información y subirla al repositorio.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git rm README.md
rm 'README.md'

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ echo "hola" > fichero

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ echo "hola" > fichero2

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ echo "hola" > fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git add *
warning: in the working copy of 'fichero', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'fichero2', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'fichero3', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git commit -m "Inicializando la rama new_master"
[new_master 694dc2d] Inicializando la rama new_master
4 files changed, 3 insertions(+), 27 deletions(-)
delete mode 100644 README.md
create mode 100644 fichero
create mode 100644 fichero2
create mode 100644 fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git log --oneline
694dc2d (HEAD -> new_master) Inicializando la rama new_master
2338057 (origin/master, master) Revision rellenando el nuevo readme
0ba469a Revision rellenando el nuevo readme
9ee248e Update pokemon-detail.component.ts
7fab61e Update pokemon-detail.component.spec.ts
67574bb Update test.ts
884d871 Update material.module.ts
8fab531 Update app.component.spec.ts
f6bb2c7 first commit

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ |
```

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git push origin new_master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 309.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'new_master' on GitHub by visiting:
remote:   https://github.com/DaniBalsera/ApiDaniFernandez/pull/new/new_master
remote:
To https://github.com/DaniBalsera/ApiDaniFernandez
 * [new branch]      new_master -> new_master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ |

```

Resultado:

The screenshot shows the GitHub interface for the repository `DaniBalsera / ApiDaniFernandez`. The repository is public. The default branch is `new_master`, which was updated 1 hour ago by DaniBalsera. A notification states: "Your master branch isn't protected. Protect this branch from force pushing or deletion, or require status checks before merging. [Learn more](#)". The "Your branches" section lists the `new_master` branch, updated 5 minutes ago by DaniBalsera.

4. Ver las ramas existentes.

- **git branch**: ver las ramas existentes a nivel local
- **git branch -a**: ver las ramas existentes a nivel local y remoto
- **git branch -a -v**: ver las ramas existentes a nivel local y remoto, además de ver el último commit realizado.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git branch
* master
  new_master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git branch -a
* master
  new_master
remotes/origin/master
remotes/origin/new_master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ git branch -a -v
* master                2338057 Revision rellenando el nuevo readme3
  new_master            694dc2d Inicializando la rama new_master
remotes/origin/master   2338057 Revision rellenando el nuevo readme3
remotes/origin/new_master 694dc2d Inicializando la rama new_master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master)
$ |
```

5. Vamos a aplicar los cambios que se realizaron a la rama new_master en master utilizando git merge.

Como podemos comprobar, los ficheros que se agregaron anteriormente en la rama new_master, se han aplicado a la rama master. ¿Esto para que sirve? Por ejemplo, imagínate que en un departamento estais 20 personas y todas tenéis que tener unas carpetas en común con la rama principal, pues se hace un **git merge** de la master a la rama de cada trabajador del departamento sin necesidad de ir creando directorios ni ficheros.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ ls
README      angular.json  package-lock.json  tsconfig.app.json  tsconfig.spec.json
README.md   e2e/          package.json        tsconfig.base.json  tslint.json
README3     karma.conf.js src/               tsconfig.json

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git merge new_master master
Updating 2338057..694dc2d
Fast-forward
 README.md | 27 -----
 fichero   | 1 +
 fichero2  | 1 +
 fichero3  | 1 +
 4 files changed, 3 insertions(+), 27 deletions(-)
 delete mode 100644 README.md
 create mode 100644 fichero
 create mode 100644 fichero2
 create mode 100644 fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ ls
README      e2e/          fichero3           package.json        tsconfig.base.json  tslint.json
README3     fichero       karma.conf.js      src/               tsconfig.json
angular.json fichero2      package-lock.json  tsconfig.app.json  tsconfig.spec.json


danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ |
```


Ahora con un push se deberían actualizar los cambios en nuestra rama master del repositorio de github.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git push origin master
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/DaniBalseira/ApiDaniFernandez
2338057..694dc2d master -> master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ |
```

master - 2 branches 0 tags Go to file Add file - Code

 **Your master branch isn't protected**
Protect this branch from force pushing or deletion, or require status checks before merging. [Learn more](#)

 DaniBalseira Inicializando la rama new_master 694dc2d 23 minutes ago 9 commits

📁 e2e	first commit	4 months ago
📁 src	Update pokemon-detail.component.ts	4 months ago
📄 .browserslistrc	first commit	4 months ago
📄 .editorconfig	first commit	4 months ago
📄 .gitignore	first commit	4 months ago
📄 README	Revision rellenando el nuevo readme	1 hour ago
📄 README3	Revision rellenando el nuevo readme3	1 hour ago
📄 angular.json	first commit	4 months ago
📄 fichero	Inicializando la rama new_master	23 minutes ago
📄 fichero2	Inicializando la rama new_master	23 minutes ago
📄 fichero3	Inicializando la rama new_master	23 minutes ago
📄 karma.conf.js	first commit	4 months ago
📄 package-lock.json	first commit	4 months ago
📄 package.json	first commit	4 months ago
📄 tsconfig.app.json	first commit	4 months ago
📄 tsconfig.base.json	first commit	4 months ago
📄 tsconfig.json	first commit	4 months ago
📄 tsconfig.spec.json	first commit	4 months ago
📄 tslint.json	first commit	4 months ago

6. Vamos a eliminar la rama new_master, ya que no la necesitamos, porque los datos se actualizaron en la rama master, además de aplicar esos cambios al repositorio remoto.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git branch -d new_master
Deleted branch new_master (was 694dc2d).
```

También vamos a eliminarla de manera remota.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git push origin --delete new_master
To https://github.com/DaniBalsera/ApiDaniFernandez
- [deleted]          new_master

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$
```

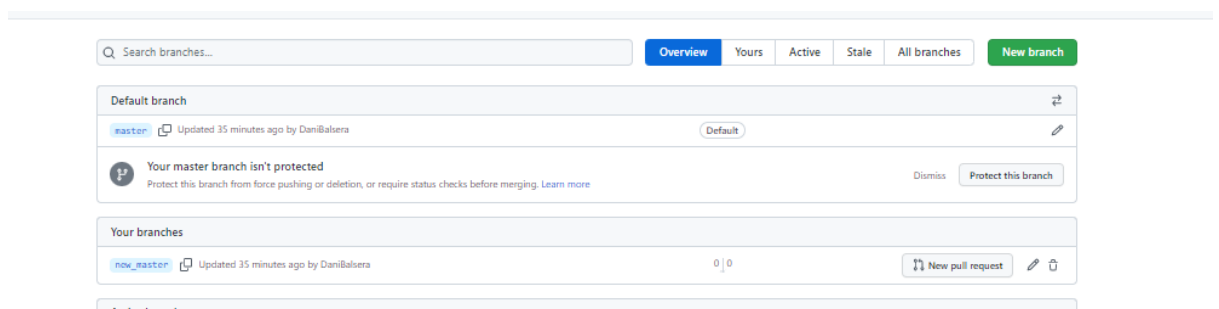
Vamos a comprobar que se ha borrado correctamente.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git branch -a -v
* master          694dc2d Inicializando la rama new_master
remotes/origin/master 694dc2d Inicializando la rama new_master

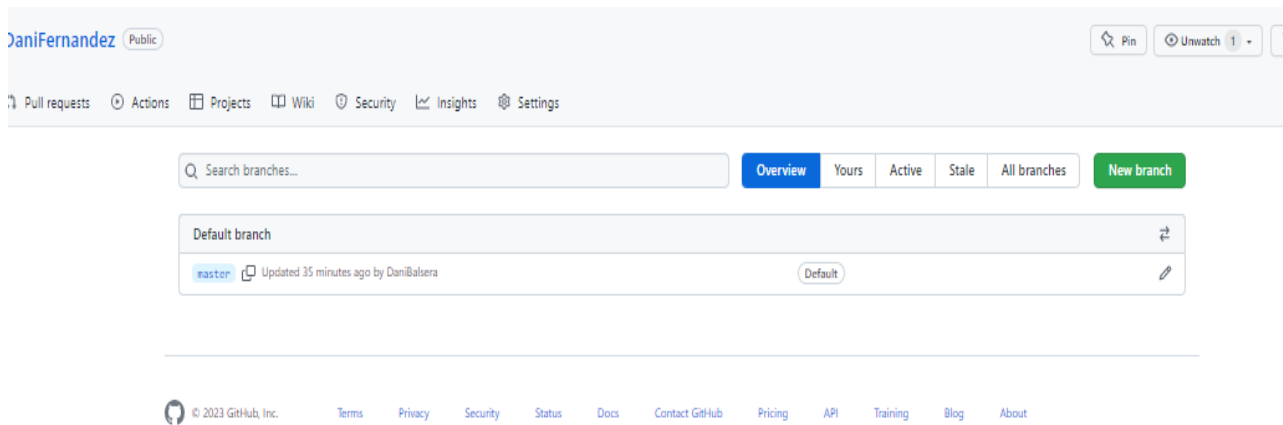
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$
```

Como podemos observar, sólo queda la rama master, que es la original. Aún así es conveniente comprobarlo también en el repositorio remoto.

Antes las ramas en el repositorio remoto estaban así:



Después de la ejecución de los comandos correspondientes:



7. Ahora vamos a ver algo interesante como puede ser crear una rama y posicionarnos en ella al mismo tiempo, sin necesidad de tener que escribir dos líneas de comandos diferentes, esto se haría de la siguiente manera:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (master)
$ git checkout -b new_master2
Switched to a new branch 'new_master2'

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario5 (new_master2)
$ |
```

-b hace referencia a branch.

De esta manera nos ahorramos tiempo en pararnos a crearla, y después desplazarnos, lo hacemos todo en una sola línea.

ENCONTRANDO ERRORES EN NUESTRO CÓDIGO

Vamos a crear un nuevo escenario para este apartado (podrías usar un único escenario para todo, pero yo por fiabilidad, estoy usando uno diferente para cada prueba).

Yo he creado 3 ficheros de prueba dentro de este repositorio:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git init
Initialized empty Git repository in C:/Users/danif/Desktop/curso_git/escenario6/.git/

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ echo "hola" >> fichero

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ echo "hola" >> fichero2

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ echo "hola" >> fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fichero
        fichero2
        fichero3

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

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git add .
warning: in the working copy of 'fichero', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'fichero2', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'fichero3', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git commit -m "Agregando nuevos ficheros al repositorio"
[master (root-commit) a5a2bc6] Agregando nuevos ficheros al repositorio
 3 files changed, 3 insertions(+)
 create mode 100644 fichero
 create mode 100644 fichero2
 create mode 100644 fichero3

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ |
```

1. Vamos a hacer cambios en un fichero, posteriormente utilizaremos la herramienta diff que utilizamos al principio del manual, para que nos muestre los cambios que está habiendo.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ echo "Encontrando errores en el escenario6" > fichero

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git diff
warning: in the working copy of 'fichero', LF will be replaced by CRLF the next time Git touch
es it
diff --git a/fichero b/fichero
index 5c1b149..c758435 100644
--- a/fichero
+++ b/fichero
@@ -1,1 @@
-hola
+Encontrando errores en el escenario6

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ |
```

2. Ahora vamos a realizar el git add .

¿No os parece curioso que justo volvemos a realizar el diff después del add y no nos muestra nada?

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git add .
warning: in the working copy of 'fichero', LF will be replaced by CRLF the next time Git touch
es it

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git diff

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ |
```

Esto sucede porque si hacemos un git diff después del git add., el fichero cambia de estado, el diff comprueba el working directory y la versión que ha sido aprobada, entonces necesitamos añadir el siguiente parámetro.

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git diff --cached
diff --git a/fichero b/fichero
index 5c1b149..c758435 100644
--- a/fichero
+++ b/fichero
@@ -1,1 @@
-hola
+Encontrando errores en el escenario6

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ |
```

Es importante recordar, que después de cada cambio hay que realizar un commit.

3. Si escribimos la siguiente línea de comandos para poder comparar cambios entre diferentes versiones de nuestros commits realizados y diferentes acciones realizadas.

El HEAD~2 significa que quiero que compare con los 2 anteriores commits, por ejemplo, hace 2 commits, el texto que estaba en mi fichero era: hola.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git diff HEAD HEAD~2
diff --git a/fichero b/fichero
index a055918..5c1b149 100644
--- a/fichero
+++ b/fichero
@@ -1,2 +1 @@
-Encontrando errores en el escenario6
+línea 3
+hola

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$
```

4. Ahora vamos a hacer una comprobación de versiones a gran escala, de manera que podamos ver incluso la fecha de los cambios y el autor de estos.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git log -p
commit 004abb2e06c1c09ca9d1de2672c3588d83642c9e (HEAD -> master)
Author: DaniBalsera <daniseneca.df@gmail.com>
Date: Sat Mar 18 18:00:07 2023 +0100

    nuevo

diff --git a/fichero b/fichero
index c758435..a055918 100644
--- a/fichero
+++ b/fichero
@@ -1 +1,2 @@
-Encontrando errores en el escenario6
+línea 3

commit 7b505cc3031d0ef08411ead13eb9d9b1e2b85e32
Author: DaniBalsera <daniseneca.df@gmail.com>
Date: Sat Mar 18 17:56:48 2023 +0100

    segundo commit

diff --git a/fichero b/fichero
index 5c1b149..c758435 100644
--- a/fichero
+++ b/fichero
@@ -1 +1 @@
-hola
+Encontrando errores en el escenario6

commit a5a2bc6aea238fe764fdbef41f6f059fe5ca6992
Author: DaniBalsera <daniseneca.df@gmail.com>
Date: Sat Mar 18 17:42:43 2023 +0100

    Agregando nuevos ficheros al repositorio

diff --git a/fichero b/fichero
new file mode 100644
index 0000000..5c1b149
--- /dev/null
+++ b/fichero
@@ -0,0 +1 @@
+hola
diff --git a/fichero2 b/fichero2
new file mode 100644
index 0000000..5c1b149
--- /dev/null
+++ b/fichero2
```

Ahora vamos a añadir un parámetro -n 2 para poder ver dos versiones hacia atrás.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario6 (master)
$ git log -p -n 2
commit 004abb2e06c1c09ca9d1de2672c3588d83642c9e (HEAD -> master)
Author: DaniBalseira <daniseneca.df@gmail.com>
Date: Sat Mar 18 18:00:07 2023 +0100

    nuevo

diff --git a/fichero b/fichero
index c758435..a055918 100644
--- a/fichero
+++ b/fichero
@@ -1,2 @@
-Encontrando errores en el escenario6
+linea 3

commit 7b505cc3031d0ef08411ead13eb9d9b1e2b85e32
Author: DaniBalseira <daniseneca.df@gmail.com>
Date: Sat Mar 18 17:56:48 2023 +0100

    segundo commit

diff --git a/fichero b/fichero
index 5c1b149..c758435 100644
--- a/fichero
+++ b/fichero
@@ -1,1 @@
-hola
+Encontrando errores en el escenario6
```

Se pueden usar parámetros para filtrar de varias formas, es tan extenso que no daría tiempo a explicar todas las formas, por lo que por aquí os dejo una documentación del comando **git log**.

<https://git-scm.com/docs/git-log>

GIT BLAME

Encontrando quien ha sido el autor de **X** movimientos en el código.

1. Nos conectamos a un repositorio remoto.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git remote add origin https://github.com/sharkdp/bat

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git fetch
remote: Enumerating objects: 14370, done.
remote: Counting objects: 100% (115/115), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 14370 (delta 50), reused 81 (delta 32), pack-reused 14255
Receiving objects: 100% (14370/14370), 29.36 MiB | 14.83 MiB/s, done.
Resolving deltas: 100% (9585/9585), done.
From https://github.com/sharkdp/bat
* [new branch]      bat-plugins -> origin/bat-plugins
```

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git pull origin master
From https://github.com/sharkdp/bat
 * branch                master      -> FETCH_HEAD
Updating files: 100% (833/833), done.

```

2. Vamos a comprobar quiénes han hecho los últimos 4 cambios.

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git log -p -n 4
commit d20405b975ee4e0927e39d4ba9ef89991a41d685 (HEAD -> master, origin/master)
Merge: 6428125 52f6239
Author: David Peter <sharkdp@users.noreply.github.com>
Date: Thu Mar 16 09:07:51 2023 +0100

    Merge pull request #2505 from nickelc/ci-msrv

    Fix 'jq' expression for retrieving 'rust-version' in MSRV build job

commit 52f6239d2870a71ac7059cde7ea473f28496891a
Author: Constantin Nickel <constantin.nickel@gmail.com>
Date: Wed Mar 15 23:11:58 2023 +0100

    Fix 'jq' expression for retrieving 'rust-version' in MSRV build job

diff --git a/.github/workflows/CICD.yml b/.github/workflows/CICD.yml
index 211815f..a7fc559 100644
--- a/.github/workflows/CICD.yml
+++ b/.github/workflows/CICD.yml
@@ -43,7 +43,7 @@ jobs:
   - name: Get the MSRV from the package metadata
     id: msrv
     run: cargo metadata --no-deps --format-version 1 | jq -r '"version=" + (.packages[] | s
select(.name == "bat").rust_version)' >> $GITHUB_OUTPUT
+   run: cargo metadata --no-deps --format-version 1 | jq -r '"version=" + (.packages[] | s
select(.name == "bat").rust_version)' >> $GITHUB_OUTPUT
   - name: Install rust toolchain (v${{ steps.msrv.outputs.version }})
     uses: dtolnay/rust-toolchain@master
     with:

commit 6428125827cf4818974e45e3aa4f8803c0436620
Merge: 231ad86 c094cd3
Author: David Peter <sharkdp@users.noreply.github.com>
Date: Wed Mar 15 22:39:03 2023 +0100

    Merge pull request #2504 from nickelc/deps/clap

    Update 'clap' to 4.1.8

commit c094cd3ee5ac35b6803b43b7ed3be391ca4b9f8b
Author: Constantin Nickel <constantin.nickel@gmail.com>
Date: Wed Mar 15 07:42:01 2023 +0100

    Update 'clap' to 4.1.8

```

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git log -n 4 --oneline
d20405b (HEAD -> master, origin/master) Merge pull request #2505 from nickelc/ci-msrv
52f6239 Fix 'jq' expression for retrieving 'rust-version' in MSRV build job
6428125 Merge pull request #2504 from nickelc/deps/clap
c094cd3 Update 'clap' to 4.1.8

```

3. Vamos a comprobar quien ha hecho cambios en un fichero concreto.

Podemos observar el nombre de usuario que realizado cambios, la fecha de los cambios, etc.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git blame src/output.rs
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200   1) use std::io::{self, Write};
014d7545 (David Tolnay         2020-03-30 10:36:26 -0700   2) #[cfg(feature = "paging")]
014d7545 (David Tolnay         2020-03-30 10:36:26 -0700   3) use std::process::Child;
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200   4)
702cb198 (sharkdp             2020-04-22 21:45:47 +0200   5) use crate::error::*;
014d7545 (David Tolnay         2020-03-30 10:36:26 -0700   6) #[cfg(feature = "paging")]
5114c018 (Nathan Fisher       2022-05-04 01:56:38 -0400   7) use crate::less::{retrieve_less_v
ersion, LessVersion};
8961f7ae (sharkdp             2020-04-22 22:54:33 +0200   8) #[cfg(feature = "paging")]
8961f7ae (sharkdp             2020-04-22 22:54:33 +0200   9) use crate::paging::PagingMode;
bbef2f41 (gahag               2020-10-07 12:28:25 -0300  10) #[cfg(feature = "paging")]
bbef2f41 (gahag               2020-10-07 12:28:25 -0300  11) use crate::wrapping::WrappingMode
;
9316f2a7 (sharkdp             2018-08-22 22:29:12 +0200  12)
53f5a37f (gahag               2020-10-07 14:50:44 -0300  13) #[cfg(feature = "paging")]
6615eceb (gahag               2020-10-07 19:14:33 -0300  14) #[derive(Debug, PartialEq)]
53f5a37f (gahag               2020-10-07 14:50:44 -0300  15) enum SingleScreenAction {
53f5a37f (gahag               2020-10-07 14:50:44 -0300  16)     Quit,
53f5a37f (gahag               2020-10-07 14:50:44 -0300  17)     Nothing,
53f5a37f (gahag               2020-10-07 14:50:44 -0300  18) }
53f5a37f (gahag               2020-10-07 14:50:44 -0300  19)
2253d073 (Fahmi Akbar Wildana 2019-10-15 08:25:53 +0700  20) #[derive(Debug)]
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  21) pub enum OutputType {
014d7545 (David Tolnay         2020-03-30 10:36:26 -0700  22)     #[cfg(feature = "paging")]
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  23)     Pager(Child),
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  24)     Stdout(io::Stdout),
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  25) }
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  26)
120b33a9 (Ezinwa Okpoechi      2018-05-21 14:59:42 +0200  27) impl OutputType {
42e3825d (David Tolnay         2020-03-30 13:18:41 -0700  28)     #[cfg(feature = "paging")]
b3903175 (sharkdp             2020-10-11 21:57:12 +0200  29)     pub fn from_mode(
b3903175 (sharkdp             2020-10-11 21:57:12 +0200  30)         paging_mode: PagingMode,
b3903175 (sharkdp             2020-10-11 21:57:12 +0200  31)         wrapping_mode: WrappingMo
```

4. También podemos especificar la línea del fichero que especifiquemos y que nos muestre los autores que han modificado esa línea.

Yo a nivel personal considero que **git blame** es una herramienta poderosa y sobretodo dentro del ámbito empresarial, ya que si por ejemplo, estáis trabajando en un proyecto y una línea de código no la entiendes o no te cuadra, puedes ver quién ha sido el autor de dichas modificaciones.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ git blame -L 6,8 src/output.rs
014d7545 (David Tolnay         2020-03-30 10:36:26 -0700 6) #[cfg(feature = "paging")]
5114c018 (Nathan Fisher       2022-05-04 01:56:38 -0400 7) use crate::less::{retrieve_less_version,
LessVersion};
8961f7ae (sharkdp             2020-04-22 22:54:33 +0200 8) #[cfg(feature = "paging")]

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/escenario7 (master)
$ |
```

REESCRIBIENDO LA HISTORIA DE COMMITS CON GIT

Vamos a utilizar git rebase.

1. Vamos a abrir un proyecto y provocar varios commits. Da igual el tipo de fichero que utilizéis, yo he creado un fichero README.md y fichero.txt.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ git log --oneline
9f6c03b (HEAD -> master) tercer commit del txt
7cfcc76 segundo commit del txt
c4e8115 primer commit del txt
ffb644b tercer commit del readme
b721391 segundo commit del readme
812cb7a primer commit del readme

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ ls -lta
total 10
drwxr-xr-x 1 danif 197609  0 Mar 18 20:13 .git/
-rw-r--r-- 1 danif 197609 45 Mar 18 20:13 fichero.txt
drwxr-xr-x 1 danif 197609  0 Mar 18 20:12 ./
-rw-r--r-- 1 danif 197609 27 Mar 18 20:12 README.md
drwxr-xr-x 1 danif 197609  0 Mar 18 20:02 ../

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ |
```

2. Imaginemos que queremos condensar todos los commits que hemos mostrado, se haría de la siguiente manera, el comando que voy a mostrar se encarga de mostrarnos una interfaz en la que decidimos que commits condensamos y cuales no. --Root significa que queremos aplicar este comando a partir del primer commit.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
git rebase --interactive --root|
```

Interfaz en cuestión:

```
C:/Users/danif/Desktop/curso_git/rebase/.git/rebase-merge/git-rebase-todo
pick 812cb7a primer commit del readme
pick b721391 segundo commit del readme
pick ffb644b tercer commit del readme
pick c4e8115 primer commit del txt
pick 7cfcc76 segundo commit del txt
pick 9f6c03b tercer commit del txt

# Rebase 9f6c03b onto a1d100e (6 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                               commit's log message, unless -C is used, in which case
```

3. Vamos a usar por ejemplo **squash** en vez de pick en algunas líneas, esto lo que hará será combinar los commits,

```
C:/Users/danif/Desktop/curso_git/rebase/.git/rebase-merge/git-rebase-todo  Modif
pick 812cb7a primer commit del readme
pick b721391 segundo commit del readme
squash ffb644b tercer commit del readme
squash c4e8115 primer commit del txt
squash 7cfcc76 segundo commit del txt
squash 9f6c03b tercer commit del txt

# Rebase 9f6c03b onto e8b5631 (6 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
```

Cuando guardemos el archivos y salgamos, nos abrirá otro en el que nos pedirá que esojamos el mensaje que vamos a utilizar para el commit del rebase.

```
GNU nano 7.1 C:/Users/danif/Desktop/curso_git/rebase/.git/COMMIT_EDITMSG
# This is a combination of 5 commits.
# This is the 1st commit message:

segundo commit del readme

# This is the commit message #2:

tercer commit del readme

# This is the commit message #3:

primer commit del txt

# This is the commit message #4:

segundo commit del txt

# This is the commit message #5:

tercer commit del txt

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sat Mar 18 20:12:16 2023 +0100
#
# interactive rebase in progress; onto e8b5631
#
# You are currently rebasing branch 'master' on 'e8b5631'.
#
# Changes to be committed:
#   modified:   README.md
#   new file:   fichero.txt
#
```

Borramos todos los mensajes y agregamos uno nuevo, además de quitar el comentario de modified para que realice dichos cambios: README.md.


```

GNU nano 7.1 C:/Users/danif/Desktop/curso_git/rebase/.git/COMMIT_EDITMSG
# This is a combination of 5 commits.
# This is the 1st commit message:

# This is the commit message #2:

# This is the commit message #3:

# This is the commit message #4:

# This is the commit message #5:

commit para el rebase que hemos elegido

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sat Mar 18 20:12:16 2023 +0100
#
# interactive rebase in progress; onto e8b5631
# Last commands done (6 commands done):
#   squash 7cfcc76 segundo commit del txt
#   squash 9f6c03b tercer commit del txt
# No commands remaining.
# You are currently rebasing branch 'master' on 'e8b5631'.
#
# Changes to be committed:
|   modified:   README.md
|   new file:   fichero.txt
#

```

Guardamos y salimos.

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ git rebase --interactive --root
[detached HEAD 4f201c2] commit para el rebase que hemos elegido
Date: Sat Mar 18 20:12:16 2023 +0100
2 files changed, 5 insertions(+)
create mode 100644 fichero.txt
Successfully rebased and updated refs/heads/master.

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ |

```

Cambios aplicados.

Con esto lo que hemos hecho es sanear el histórico de dicho proyecto, para que no se acumulen los diferentes commits.

```

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ git log --oneline
4f201c2 (HEAD -> master) commit para el rebase que hemos elegido
812cb7a primer commit del readme

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/rebase (master)
$ |

```

VAMOS A TRABAJAR CON ETIQUETAS DE CÓDIGO

En este apartado vamos a ver como darle un nombre a la versión de nuestro código.

Tipos de etiquetas:

- Etiquetas ligeras
- Etiquetas anotadas (Son las que albergan más información)

1. Estado actual del proyecto y sus commits.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ ls -l
total 10
drwxr-xr-x 1 danif 197609  0 Mar 19 19:57 ../
-rw-r--r-- 1 danif 197609 27 Mar 19 19:59 README.md
drwxr-xr-x 1 danif 197609  0 Mar 19 19:59 ./
-rw-r--r-- 1 danif 197609 15 Mar 19 19:59 fichero.txt
drwxr-xr-x 1 danif 197609  0 Mar 19 19:59 .git/

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ git log --oneline
27e9c0c (HEAD -> master) primer commit del texto
dc18726 tercer commit del readme
f901fb3 segundo commit del readme
23b2fed primer commit del readme

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ |
```

2. Para que os hagáis una idea, lo que hemos visto en la captura anterior es la versión actual de nuestro código, imaginemos que a todo eso le queremos asignar una etiqueta especificando la versión que es, se tendría que realizar de la siguiente manera:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ git tag -a v1.5 -m "Esta es la version 1.5"

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ |
```

3. Ahora queremos saber toda la información sobre la versión del código sobre el que estamos trabajando, se tendría que realizar de la siguiente manera:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ git show v1.5
tag v1.5
Tagger: DaniBalseira <daniseneca.df@gmail.com>
Date: Sun Mar 19 20:01:04 2023 +0100

Esta es la version 1.5

commit 27e9c0c51af5a733cc7ba1cc414d493a7203cd36 (HEAD -> master, tag: v1.5)
Author: DaniBalseira <daniseneca.df@gmail.com>
Date: Sun Mar 19 19:59:48 2023 +0100

    primer commit del texto

diff --git a/fichero.txt b/fichero.txt
new file mode 100644
index 0000000..a0d8b0c
--- /dev/null
+++ b/fichero.txt
@@ -0,0 +1 @@
+prueba 1 texto

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$
```

4. Ahora imaginemos que tenemos un proyecto grande (voy a conectarme remotamente a uno), y queremos saber lo que ha pasado en una versión específica.

Con fetch podemos ver todas las versiones de un proyecto.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ git fetch
remote: Enumerating objects: 14370, done.
remote: Counting objects: 100% (115/115), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 14370 (delta 50), reused 81 (delta 32), pack-reused 14255
Receiving objects: 100% (14370/14370), 29.36 MiB | 16.83 MiB/s, done.
Resolving deltas: 100% (9585/9585), done.
From https://github.com/sharkdp/bat
* [new branch]      bat-plugins      -> origin/bat-plugins
* [new branch]      ci-experiment    -> origin/ci-experiment
* [new branch]      create_highlighted_versions-wrong-path-repro -> origin/create_highlighted_versions-wrong-path-repro
* [new branch]      crontab          -> origin/crontab
* [new branch]      master           -> origin/master
* [new branch]      release-v0.18.3  -> origin/release-v0.18.3
* [new branch]      sponsor          -> origin/sponsor
* [new tag]         v0.18.3          -> v0.18.3
* [new tag]         v0.1.0           -> v0.1.0
* [new tag]         v0.10.0          -> v0.10.0
* [new tag]         v0.11.0          -> v0.11.0
* [new tag]         v0.12.0          -> v0.12.0
* [new tag]         v0.12.1          -> v0.12.1
* [new tag]         v0.13.0          -> v0.13.0
* [new tag]         v0.14.0          -> v0.14.0
* [new tag]         v0.15.0          -> v0.15.0
* [new tag]         v0.15.1          -> v0.15.1
* [new tag]         v0.15.2          -> v0.15.2
* [new tag]         v0.15.3          -> v0.15.3
* [new tag]         v0.15.4          -> v0.15.4
* [new tag]         v0.16.0          -> v0.16.0
* [new tag]         v0.17.0          -> v0.17.0
```

Ahora queremos consultar que ha pasado en la versión v0.1.0, pues se realizaría de la siguiente manera.

Así podremos consultar siempre la información que necesitemos sobre una versión específica, por ejemplo se podría aplicar en el caso que una versión de la aplicación que vuestro que equipo de trabajo ha hecho tiene errores, pues de esta manera podríais obtener información de dicha versión.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ git show v0.1.0
commit 9435e16a1be6e7fab41a067284d707217b42afb5 (tag: v0.1.0)
Author: sharkdp <davidpeter@web.de>
Date:   Sun Apr 22 16:21:58 2018 +0200

    Update .travis.yml

diff --git a/.travis.yml b/.travis.yml
index 02456b7..8f5463e 100644
--- a/.travis.yml
+++ b/.travis.yml
@@ -32,7 +32,7 @@ env:
     # Used as conditional check in the install stage
     - HOST=x86_64-unknown-linux-gnu
     # Used on the deployment script
-    - PROJECT_NAME=hyperfine
+    - PROJECT_NAME=bat

install:
    # prevent target re-add error from rustup

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/etiquetas (master)
$ |
```

MOVERSE ENTRE LOS ESTADOS DE UN REPOSITORIO GIT

En este apartado aprenderemos cómo poder usar git para reflejar cada uno de los estados que ha tenido nuestro proyecto, y podamos acceder a **x** estado para poder actuar por ejemplo para ciertas modificaciones.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git log --oneline
756c25d (HEAD -> master) primer commit del fichero de texto 2
64b9e31 primer commit del fichero de texto
77a0843 primer commit del README

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
```

Imaginemos que todos los cambios anteriores se han efectuado en diferentes días.

1. Vamos a movernos en el tiempo, por ejemplo creamos dos ficheros.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ echo "mi fichero 3" >> prueba1

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ echo "mi fichero 4" >> prueba2
```

2. Uno de ellos lo vamos a agregar al staging.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git add prueba1
warning: in the working copy of 'prueba1', LF will be replaced by CRLF the next time Git touch
es it
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ |
```

3. Ahora que ocurre, básicamente se nos ha olvidado agregar la prueba2 antes del commit , por lo que haremos un commit --amend para actualizar los cambios y que se aplique un nuevo commit de manera mas automatizada. Es decir, estamos moviéndonos al estado donde se ha generado el anterior commit para que prueba2 se meta dentro de staging con el commit correspondiente a prueba1 también.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git commit --amend
```

Lo redondeado estaba comentado, hay que descomentarlo.

Arriba ponemos la nueva linea de commit, antes era únicamente **agrego prueba 1**.

```
GNU nano 7.1 C:/Users/danif/Desktop/curso_git/estados/.git/COMMIT_EDITMSG
agrego prueba 1 y prueba 2 (actualizado)

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Mar 19 20:17:38 2023 +0100
#
# On branch master
# Changes to be committed:
#   new file:   prueba1
#   new file:   prueba2
#
```

Guardamos el archivo y realizamos un git log.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git log --oneline
4e24f3c (HEAD -> master) agrego prueba 1 y prueba 2 (actualizado)
756c25d primer commit del fichero de texto 2
64b9e31 primer commit del fichero de texto
77a0843 primer commit del README
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
```

Ya tenemos el commit actualizado.

Ahora supongamos que queremos saber como nos encontrábamos en este punto:

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git log --oneline
4e24f3c (HEAD -> master) agrego prueba 1 y prueba 2 (actualizado)
756c25d primer commit del fichero de texto 2
64b9e31 primer commit del fichero de texto
77a0843 primer commit del README
```

Realizamos lo siguiente:

Ponemos el Hash ID del commit y realizamos un checkout, esto lo que hace es movernos al estado en el que se encontraba nuestro proyecto cuando únicamente habíamos realizado el commit del README, por lo que si hacemos un ls , únicamente nos va a mostrar el README.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados (master)
$ git checkout 77a0843
Note: switching to '77a0843'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 77a0843 primer commit del README

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados ((77a0843...))
$ ls -l
total 1
-rw-r--r-- 1 danif 197609 9 Mar 19 20:13 README.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/estados ((77a0843...))
$ |
```

HACIENDO COPIA DE NUESTRO PROYECTO DE MANERA TEMPORAL CON GIT STASH

Estado actual de nuestro repositorio.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ echo "una linea para readme.md" > README.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Gi
ches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ git commit -m "IC"
[master 9e684a5] IC
1 file changed, 1 insertion(+), 1 deletion(-)

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
```

1. Supongamos que agregamos una nueva linea al README.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ echo "agrego una nueva linea" >> README.md

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git tou
ches it

danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

2. Realizamos un git stash para hacer una foto del estado actual.

```
danif@DESKTOP-H4UMD08 MINGW64 ~/Desktop/curso_git/stash (master)
$ git stash
Saved working directory and index state WIP on master: 9e684a5 IC
```

3. Ahora vamos a eliminar el README y vamos a volver a crearlo para crear una colisión, por lo que el stash detectará dicha colisión, y usaremos stash apply para poder usar la foto que hicimos anteriormente y agregarlo con un git add .

```
danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ git rm README.md
rm 'README.md'

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ echo "nuevo contenido" > README.md

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ git stash apply
error: The following untracked working tree files would be overwritten by merge:
    README.md
Please move or remove them before you merge.
Aborting
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ git add README.md
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it

danif@DESKTOP-H4UMDO8 MINGW64 ~/Desktop/curso_git/stash (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md
```

Esto ha sido todo, es un manual básico guiado que creo que os podrá venir bien de cara a las empresas, pero el mundo de git es mucho más amplio, os dejo aquí toda la documentación de git:

<https://git-scm.com/doc>