

ETS-UT5-Git y Github

Repaso. Configuración inicial de Git

Después de instalar `git` en nuestro equipo debemos realizar la configuración inicial de los parámetros de nuestro perfil de usuario:

```
$ git config --global user.name "your-name"
$ git config --global user.email "your-email"
```

comprobamos con:

```
$ git config --list
```

Si tenemos una carpeta en la que tenemos archivos de los que queremos hacer seguimiento mediante `git` debemos acceder a la misma e iniciar un repositorio:

```
$ cd test-git
$ git init
```

Si hemos creado o modificado ficheros podemos añadirlos para seguimiento ejecutando:

```
$ echo Hello world > first.txt
$ git add .
$ git status
On branch master
```

```
No commits yet
```

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   first.txt
```

Se está seguimiento de los ficheros y están en el área de seguimiento (`staging area`)

Si queremos que los cambios se hagan permanentes debemos realizar una operación con los ficheros que estan en el area de seguimiento `staging area`

```
$ git commit -m "first commit"
[master (root-commit) 317b1a2] initial commit
1 file changed, 1 insertion(+)
create mode 100644 first.txt
```

La opción `-m` nos permite especificar un breve mensaje explicando los cambios realizados.

Una forma de elaborar buenos mensajes para los `commits` es pensar en como completar la frase si se aplica, este commit consigue que ...

El mensaje debería explicar el motivo del cambio; explicamos el por qué, no el cómo.

git branches

A continuación, vamos a simular un flujo de trabajo que se puede dar mientras trabajamos en un proyecto.

Si vamos a añadir una nueva funcionalidad al proyecto debemos crear una rama para no interferir con el trabajo de los compañeros:

Para crear una rama ejecutamos de nombre `feature-name` :

```
$ git branch feature-info
```

Podemos comprobar que se ha creado:

```
$ git branch
  feature-info
* master
```

Se muestran las ramas existentes. El `*` indica cual es la rama de trabajo actual.

Para cambiarnos a dicha rama lo podemos hacer con

```
$ git switch feature-info
```

Hacemos las modificaciones:

```
$ echo Testing git > info.txt
$ git add .
$ git commit -m "crear fichero con info"
[feature-info 9e73e6a] crear fichero con info
1 file changed, 1 insertion(+)
create mode 100644 info.txt
```

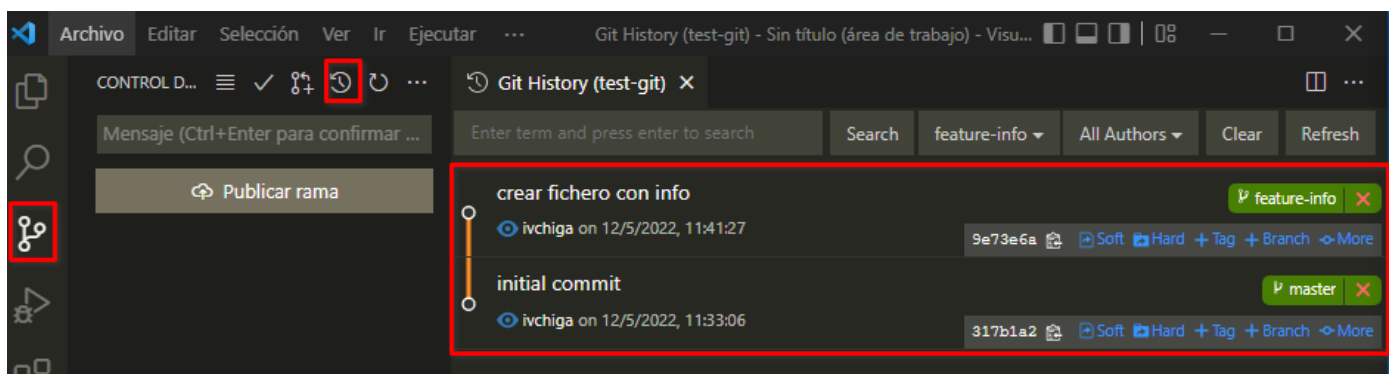
Para ver el historial de los cambios realizados podemos ejecutar:

```
$ git log --oneline
9e73e6a (HEAD -> feature-info) crear fichero con info
317b1a2 (master) initial commit
```

A tener en cuenta:

- Los cambios se muestran en orden inverso. La primera línea muestra el último cambio realizado.
- Las columnas son:
 - hash o identificador del commit
 - rama en la que se realizó el cambio
 - mensaje del commit
- **HEAD** apunta en el repositorio a la ubicación actual en el mismo.

Si instalamos en **VSCODE** el plugin **Git History** podemos ver de forma gráfica el historial de cambios.



Supongamos que después de crear una rama surge un error en la rama principal que queremos solucionar. Pasamos a la rama principal:

```
$ git switch master
```

Creamos una rama para resolver el error y cambiamos directamente a la misma con:

```
$ git switch -c bug-fix # crea y cambia a la rama
Switched to a new branch 'bug-fix'
```

Resolvemos el error:

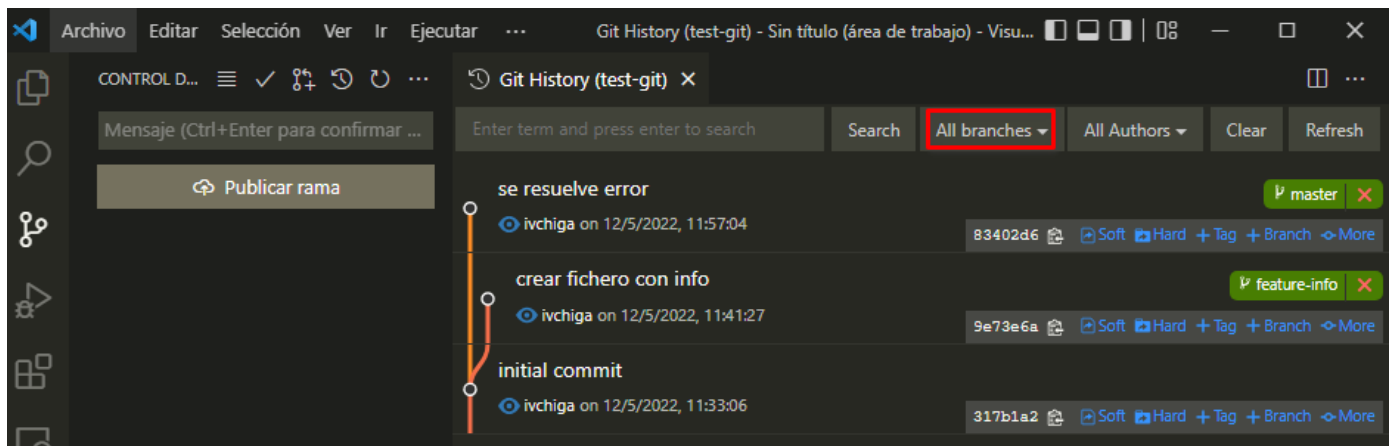
```
$ echo Hola a todos > first.txt
$ git add .
$ git commit -m "se resuelve error"
```

Una vez resuelto el error podemos volver a la rama principal y fusionar el código de la rama bug-fix ejecutando:

```
$ git switch master
$ git merge bug-fix
Updating 317b1a2..83402d6
Fast-forward
 first.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Al no ser necesaria ya la rama la podemos eliminar:

```
$ git branch -d bug-fix
```



Si queremos volver a la rama en la que estabamos trabajando y añadir funcionalidad:

```
$ git switch feature-info
$ echo Mis mejores deseos >> first.txt
$ git add .
$ git commit -m "Se añade línea a first.txt"
[feature-info 47c2443] Se añade línea a first.txt
 1 file changed, 1 insertion(+)
```

Al tratar de añadir los cambios realizados a la rama principal:

```
$ git switch master
$ git merge feature-info
Auto-merging first.txt
CONFLICT (content): Merge conflict in first.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Al intentar mezclar las modificaciones de la rama se muestra que hay un conflicto porque uno de los ficheros (`first.txt`) que hemos modificado en la nueva rama, también lo modificamos en la rama `master` posteriormente a que creáramos la rama (al resolver el error que se produjo).

Para ver los detalles:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  new file:   info.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   first.txt
```

Debemos editar el fichero en el que se ha producido el conflicto.

```
1 | $ nano first.txt
```

Incluirá marcas en las partes del mismo que hay conflicto.

```
<<<<<< HEAD
Hola a todos
=====
Hello world
Mis mejores deseos
>>>>>> feature-info
```

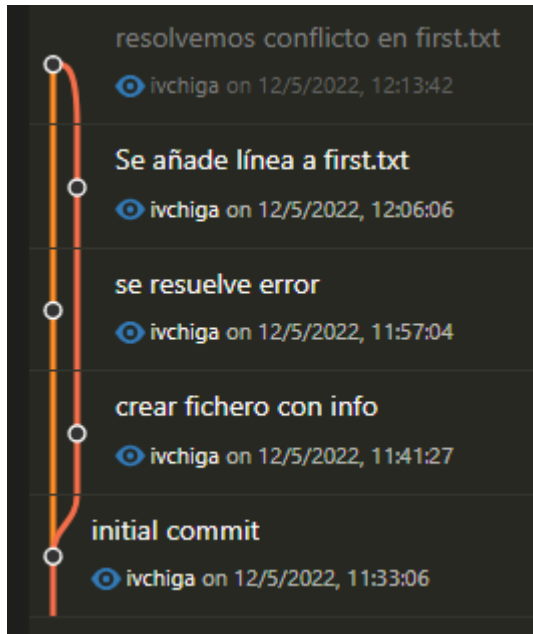
Modificamos el fichero con la versión final con la que nos queremos quedar.

```
Hola a todos
Mis mejores deseos
```

Hacemos `commit` del cambio e intentamos nuevamente fusionar la rama.

```
$ git add .  
$ git commit -m "resolvemos conflicto en first.txt"  
$ git merge feature-info
```

Si hemos resuelto el conflicto no debería producirse error y la rama está fusionada:




Ya podemos eliminar la rama:

```
$ git branch -d feature-info  
$ git branch  
* master
```

Github para proyecto en grupo

Creando un repositorio

Para empezar, uno de los miembros creamos un repositorio:



Search or jump to...

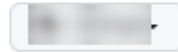
[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *



Repository name *



/ 

Great repository names are short and memorable. Need inspiration? How about [psychic-eureka?](#)

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.](#)

.gitignore template: None ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

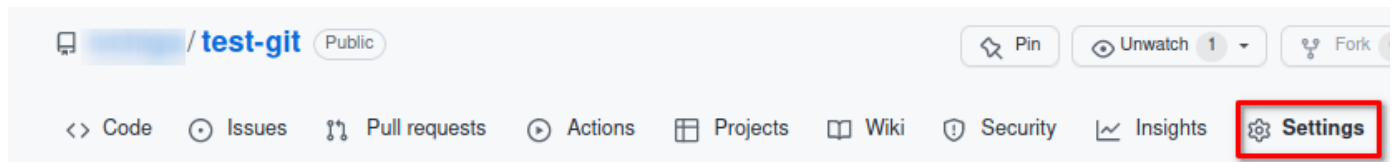
This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Añadiendo colaboradores

Añadimos como colaboradores del repositorio al resto de usuarios del grupo. En la configuración del repositorio accedemos a colaboradores:



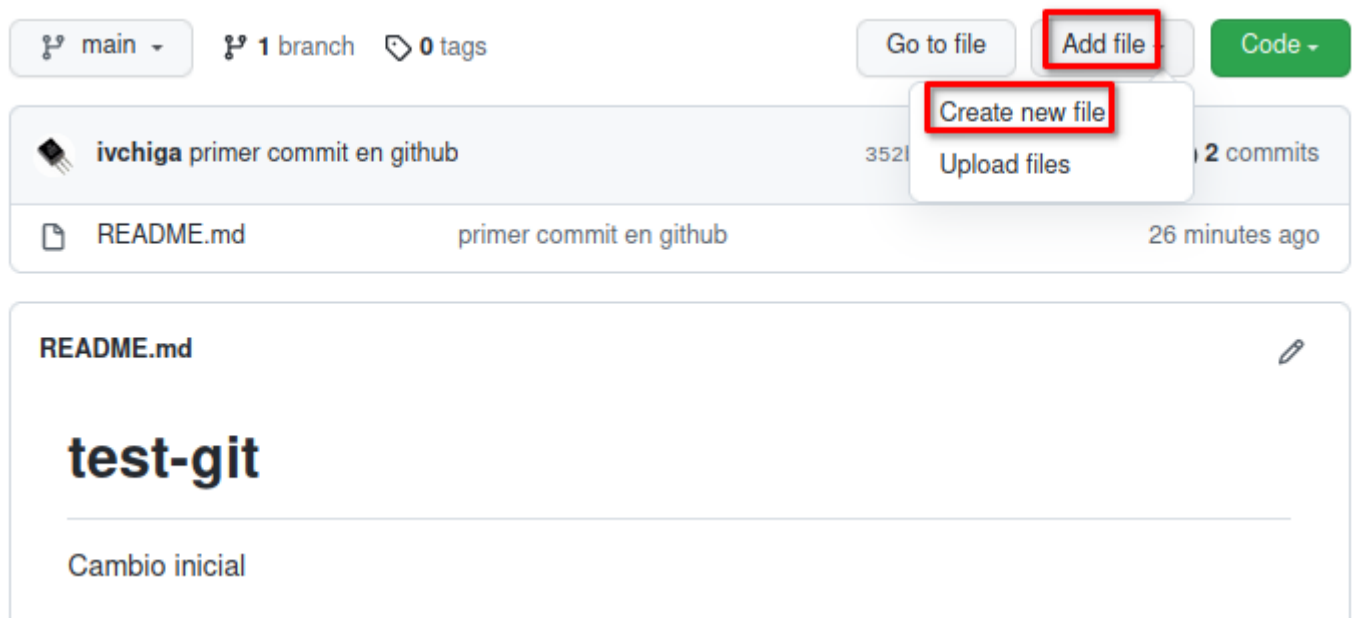
Enviamos invitaciones.

El usuario recibirá un correo. Con la sesión iniciada en Github en el navegador hacemos click en la invitación para aprobarla.

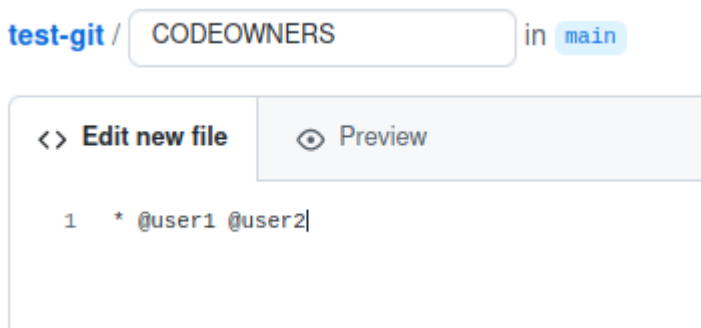
Una vez aceptada tendremos permiso para colaborar en el repositorio.

Permiso para aprobar cambios

Si queremos que uno o varios de los usuarios puedan aprobar cambios en el repositorio hemos de darles privilegio de `CODEOWNER`. Para ello creamos en el fichero raíz del repositorio un fichero de nombre `CODEOWNERS`



E introducimos en el mismo los usuarios con permiso con el siguiente formato:

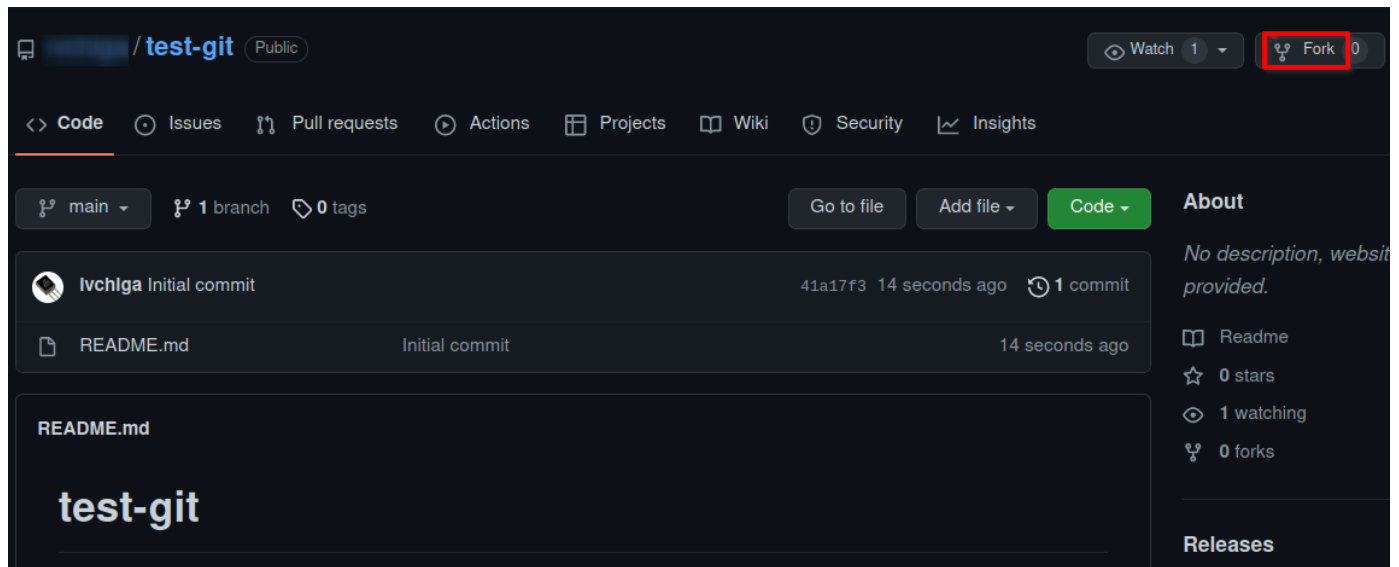


Debemos hacer commit para que se aplique el cambio.

Haciendo fork del proyecto

Para que los usuarios que recibieron la invitación puedan trabajar en el proyecto primero deben hacer un `fork` del mismo.

Con la cuenta de usuario iniciada en Github accede a la URL del proyecto y haz click en **fork**



Al completar la operación se creará una copia del repositorio en la cuenta del usuario añadido como colaborador.

Trabajando localmente

Para trabajar localmente en el repositorio hemos de sincronizarlo en nuestro equipo. Obtenemos su dirección:

Y en la carpeta en la que queremos descargar el repositorio ejecutamos:

```
$ git clone https://github.com/usuario/test-git.git
$ cd test-git
```

Nota: el usuario que creo el repositorio clonará el repositorio principal y el resto de usuarios obtendrá la dirección para clonarlo del **fork** del proyecto que tiene en su cuenta.

Github flow

Existen diferentes **metodologías** a la hora de trabajar con repositorios en proyectos. Para nuestro proyecto utilizaremos el flujo de trabajo recomendado por Github:

- Crear localmente una nueva rama - `git switch -c update-readme`

- Hacer cambios y commits
- Subir rama al repositorio
- Abrir una Pull Request - `git push origin update-readme`
- Revisar y comentar
- Hacer Merge

Antes de ver como trabajar con ramas en Github hemos de tener en cuenta que después de haber clonado el repositorio o durante el ciclo del vida del proyecto pueden haber cambios en el repositorio hechos por otros miembros del equipo que debemos sincronizar. Lo podemos hacer de dos maneras:

1. Descargar todos los commits y fusionarlos con la rama `main` local:

```
$ git fetch origin
$ git merge origin/master
```

2. Hacer la operación completa de una sola vez mediante `git pull`

```
$ git pull origin      # fetch + merge
```

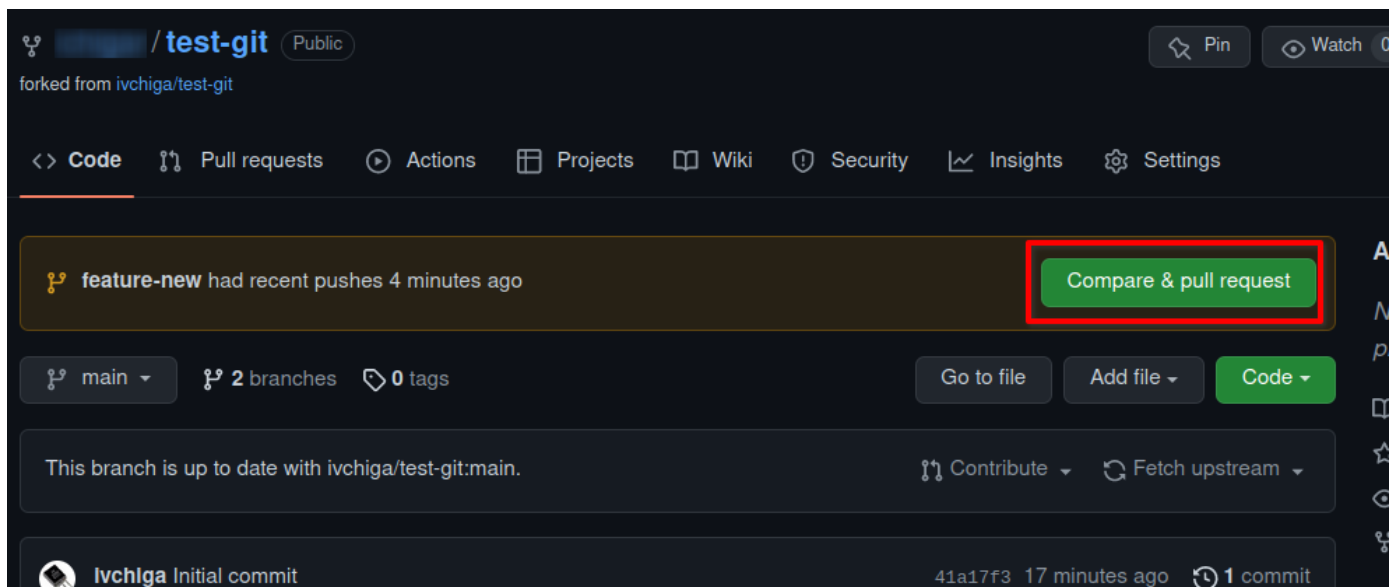
Si creamos localmente una rama y hacemos cambios:

```
$ git switch -c feature-new
$ echo nueva funcionalidad > feature.txt
$ git add .
$ git commit -m "nueva funcionalidad"
[feature-new 986e6ce] nueva funcionalidad
1 file changed, 1 insertion(+)
create mode 100644 feature.txt
```

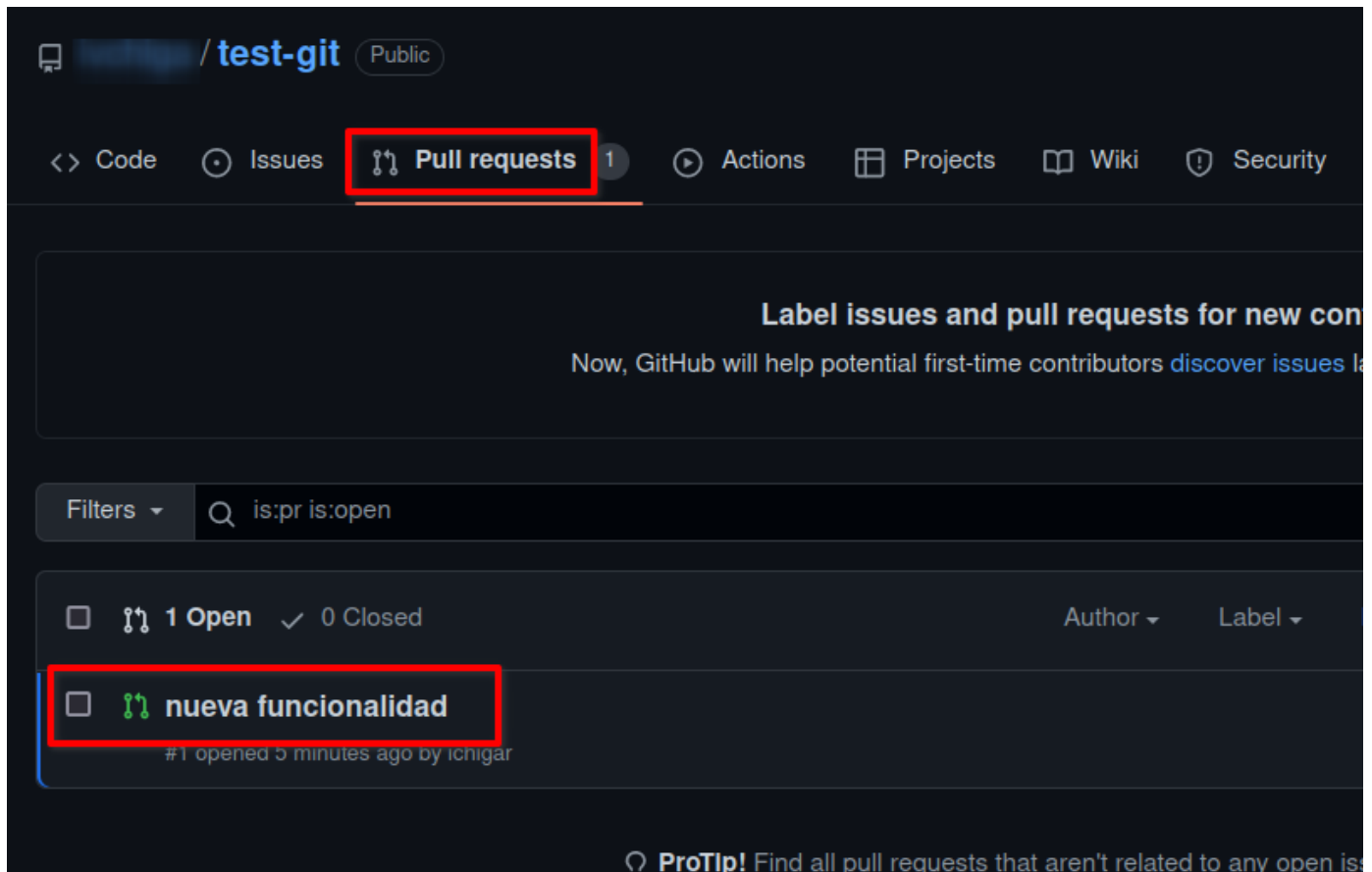
y los queremos subir al repositorio deberemos hacer una operación `push`

```
$ git push -u origin feature-new
git: 'credential-manager' no es un comando de git. Mira 'git --help'.
git: 'credential-manager' no es un comando de git. Mira 'git --help'.
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 307 bytes | 307.00 KiB/s, listo.
Total 3 (delta 0), reusado 0 (delta 0)
remote:
remote: Create a pull request for 'feature-new' on GitHub by visiting:
remote:   https://github.com/ichigar/test-git/pull/new/feature-new
remote:
To https://github.com/ichigar/test-git.git
 * [new branch]      feature-new -> feature-new
Rama 'feature-new' configurada para hacer seguimiento a la rama remota 'feature
```

Vemos que se ha creado una **pull-request**. Si vamos a nuestro repositorio veremos que se ha subido la nueva rama y se nos da la opción de dejar un comentario y crear una **Pull request**



Se recibirá en el repositorio principal del proyecto

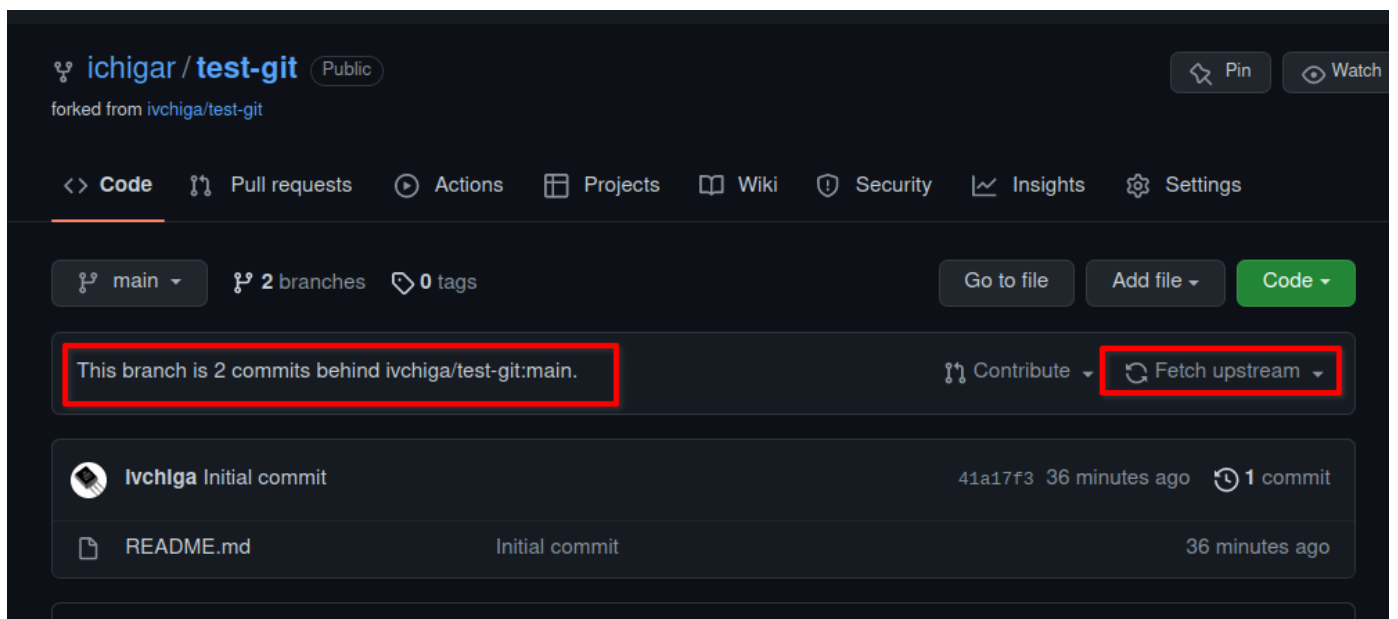


Desde allí la podremos contestar en un comentario a la petición y aprobar o denegar mezclar los cambios en la rama principal.

Después de revisar a fondo los cambios si vemos que no hay problema podemos confirmar que queremos mezclar la rama con la principal. Se aplicarán los cambios y la rama será eliminada.

Nota: Podrá hacer la operación tanto si somos el propietario del repositorio o hemos sido añadidos al fichero **CODEOWNERS**

Si vamos a la web del **fork** veremos que los cambios del repositorio no se han sincronizado.



Hacemos **Fetch upstream**

Y sincronizamos localmente los cambios.

```
$ git pull
```

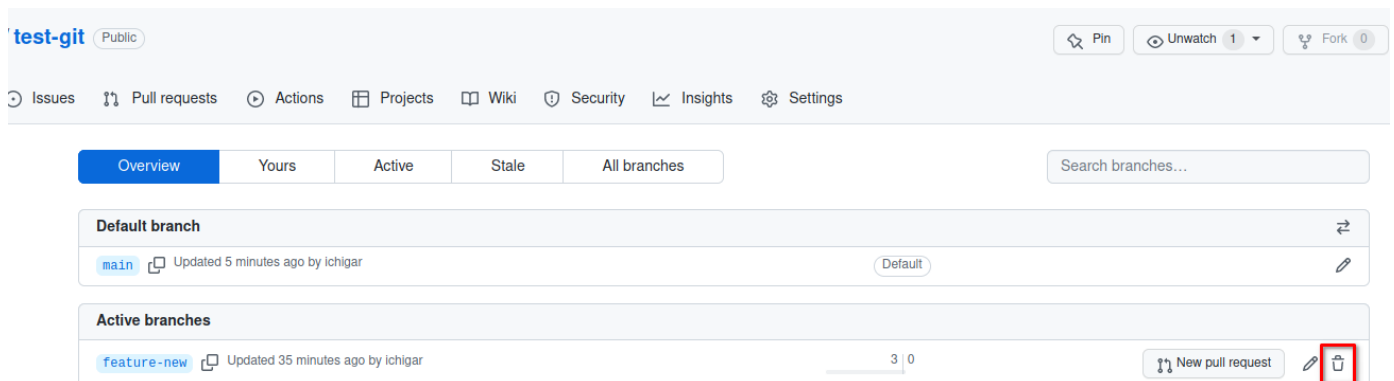
En el **fork** y localmente no se elimina la nueva rama. Cómo ya no la necesitamos la podemos eliminar.

```
$ git switch main  
$ git branch -d feature-main  
$ git push
```

Si no necesitamos más la rama la podemos eliminar:

```
$ git branch -d feature-new
```

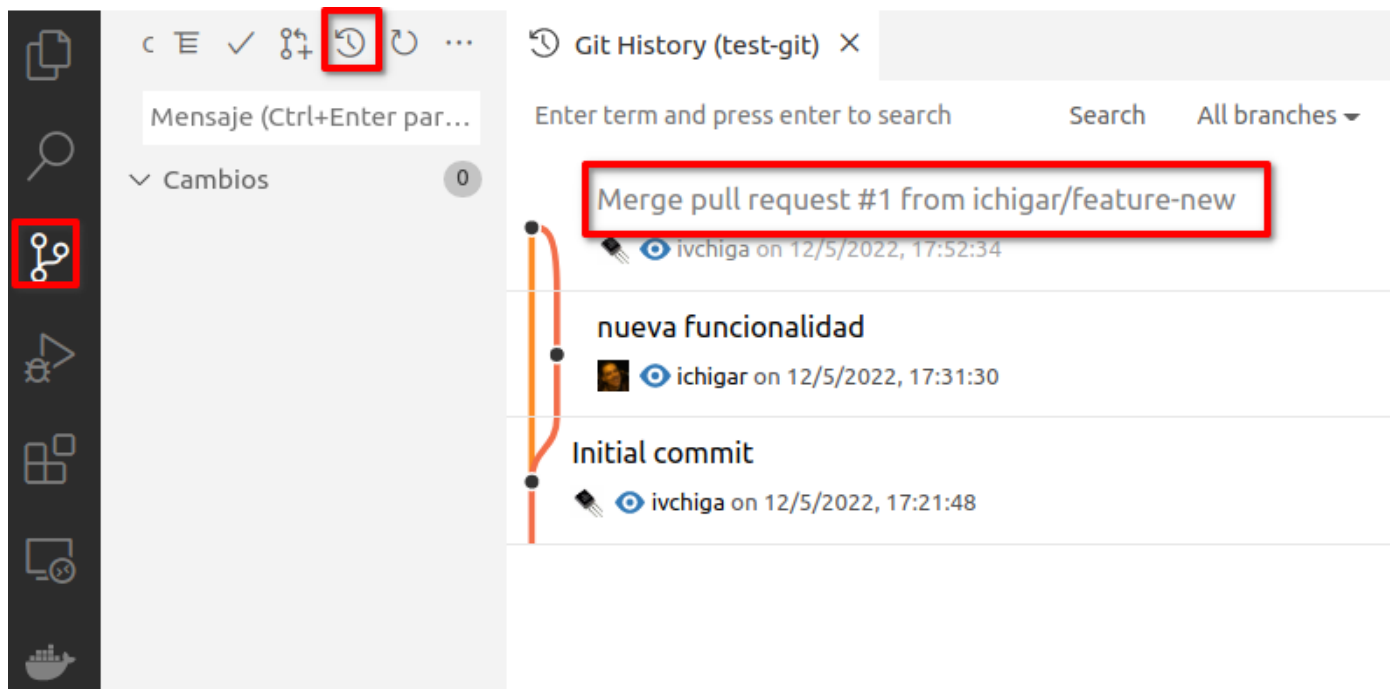
La rama no se eliminará en Github. Lo podemos hacer desde la web del repositorio



Y también podemos eliminar la rama remota desde la línea de comandos.:

```
git push origin --delete feature-new
```

Si tenemos instalado el plugin **git history** podemos ver de forma gráfica el historial de cambios aplicados.



Actividad (ver y entregar en CAMPUS)

1. Uno de los miembros del grupo creará un repositorio de nombre `test-githubflow`
2. Añadir como colaborador a cada uno de los miembros del grupo y al profesor (@ichigar)
3. Se añade a todos los colaboradores como **CODEOWNERS** del repositorio.
4. Colaboradores hacen **fork** del repositorio
5. Cada uno de los miembros del grupo crea una rama `feature-nombre` dónde nombre es el nombre del usuario.
6. Cada miembro del grupo crea en la rama un archivo `nombre.txt` dónde nombre es el nombre del usuario.
7. Dentro del archivo se añade una línea con un texto de prueba y se hace commit del mismo.
8. Cada usuario genera un **pull request**
9. Se aprueban los **pull request** y se añaden finalmente todos los archivos a la rama `main` del repositorio principal.
10. Se sincronizan los cambios en los **fork** de cada usuario.
11. Se eliminan las rama `feature-nombre` localmente y en los **fork**
12. Comprueba en el historial de cambios del repositorio que se ven todas las ramas con las aportaciones de cada usuario

Entrega en la actividad de CAMPUS el resultado

Git Undo

Mas detalles en [el siguiente artículo de freecodecamp](https://www.freecodecamp.org/espanol/news/la-guia-definitiva-para-git-reset-y-git-revert/)

(<https://www.freecodecamp.org/espanol/news/la-guia-definitiva-para-git-reset-y-git-revert/>).

Recursos

- [w3schools.com: Tutorial de Git](https://www.w3schools.com/git/default.asp) (https://www.w3schools.com/git/default.asp).
- [w3schools - github flow](https://www.w3schools.com/git/git_github_flow.asp?remote=github) (https://www.w3schools.com/git/git_github_flow.asp?remote=github).
- [w3schools - git undo](https://www.w3schools.com/git/git_revert.asp?remote=github) (https://www.w3schools.com/git/git_revert.asp?remote=github).
- [microsoft.com - github basics](https://github.com/microsoft/Web-Dev-For-Beginners/blob/main/1-getting-started-lessons/2-github-basics/README.md) (https://github.com/microsoft/Web-Dev-For-Beginners/blob/main/1-getting-started-lessons/2-github-basics/README.md).
- [Tutorial de Github para trabajo en grupo](https://www.softwaretestinghelp.com/github-tutorial/) (https://www.softwaretestinghelp.com/github-tutorial/).

tags: ets proyecto git github branches ramas pull request conflict