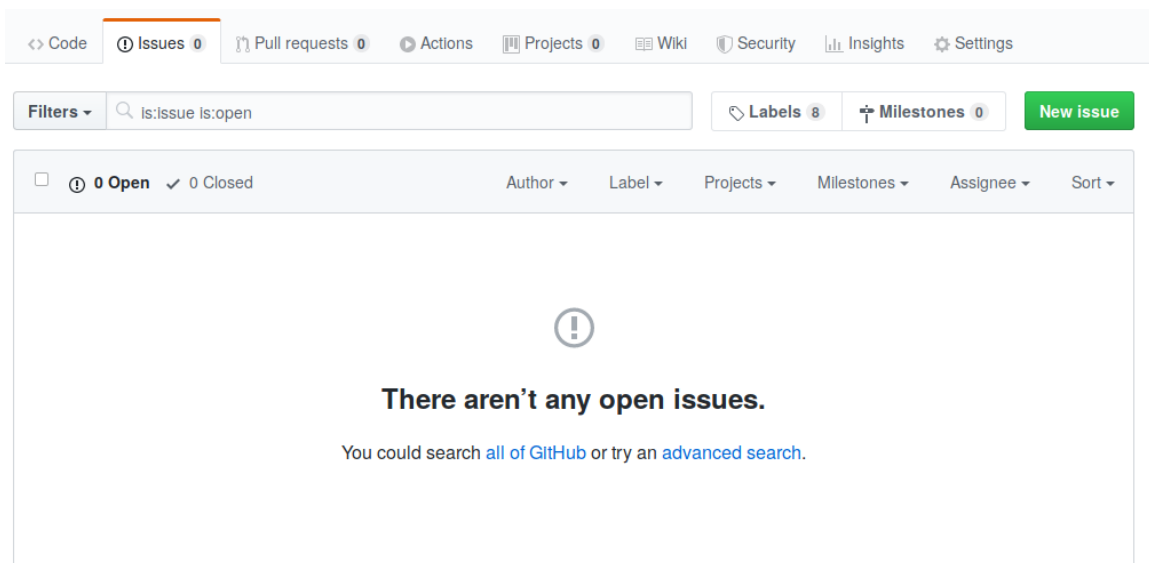


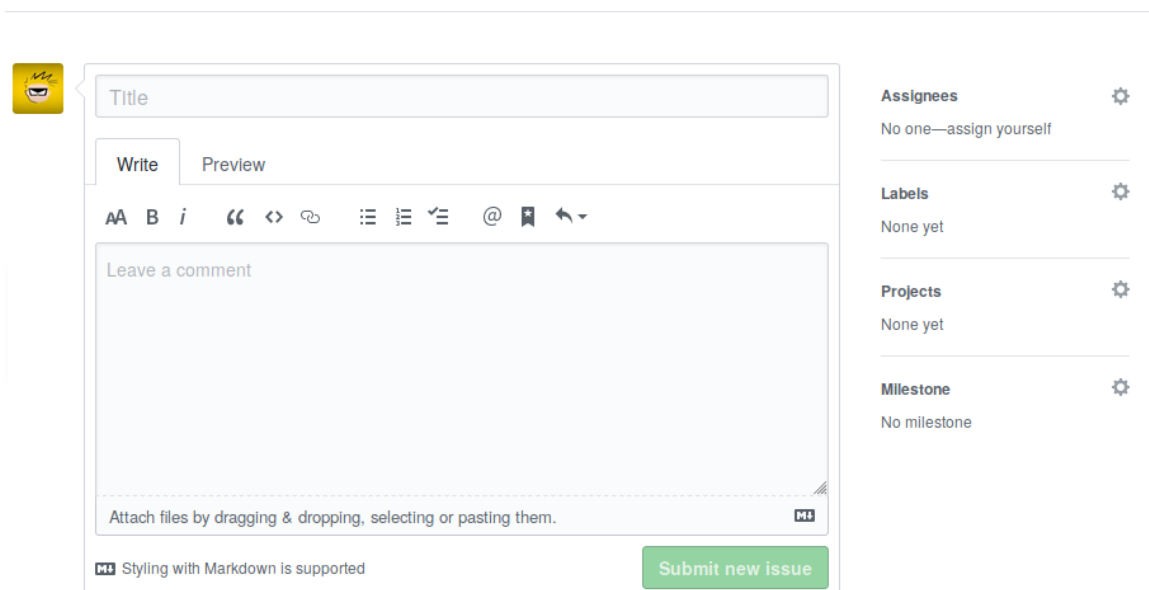
Flujo de trabajo en GitHub

Paso 0. Abrir una incidencia (issue)


Habitualmente el trabajo puede partir a raíz de un reporte por parte de un miembro del equipo o de una persona externa. Para eso tenemos la sección *Issues*.



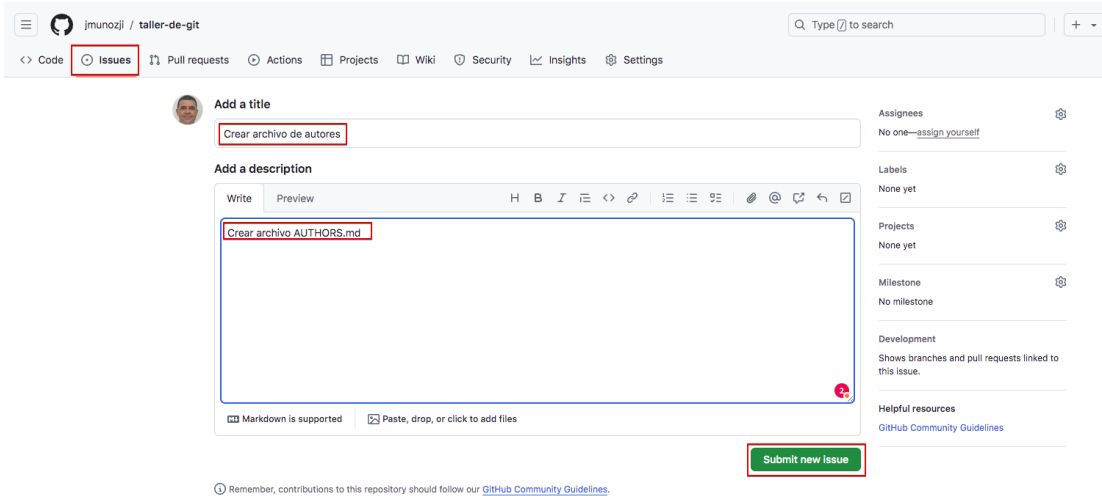
Una issue cuando se crea se compone de un título y una descripción en Markdown. Si la persona es miembro del equipo, opcionalmente puede asignarle una serie de metadatos: etiquetas (labels), hitos (milestone), proyecto al que pertenece o responsables encargados de cerrar la incidencia.



Una vez creado, al mismo se le asignará un número.

 **Example**

Vamos a crear una incidencia llamada "Crear archivo de autores", donde indiquemos que vamos a crear un archivo `AUTHORS.md` con la lista de desarrolladores del proyecto.



Paso 1. Crear una rama

Crearemos una rama cada vez que queramos implementar una nueva característica al proyecto que estamos realizando. La misma puede estar provocada por una incidencia o no.

Tip

Es una buena costumbre crear en Issues el listado de casos de uso, requisitos, historias de usuario o tareas (como lo queramos llamar), para tener un registro del trabajo que llevamos y el que nos queda.

El nombre de la rama puede ser el que creamos conveniente, pero hay que intentar ser coherente y usar siempre el mismo método, sobre todo si trabajamos en equipo.

Un método puede ser el siguiente (¡No lo hagas, es solo un ejemplo!):

```
$ # tipo-número/descripción
$ git checkout -b feature-1/create-changelog
$ git checkout -b hotfix-2/updated-database
```

En entornos de trabajo multiusuario se puede usar el siguiente (¡No lo hagas, es solo un ejemplo!):

```
$ # usuario/tipo-número/descripción
$ git checkout -b sgomez/feature-1/create-changelog
$ git checkout -b sgomez/hotfix-2/updated-database
```

De esa manera, podemos seguir fácilmente quién abrió la rama, en qué consiste y a qué *issues* está conectada. Pero como decimos es más un convenio que una imposición, pudiéndole poner el nombre que queramos.

Vamos a crear la rama y los commits correspondientes y subir la rama con push al servidor.

```
$ git checkout -b sgomez/feature-1/create-changelog
```

Ahora creamos el fichero AUTHORS.md

```
$ git add AUTHORS.md
$ git commit -m "Añadido fichero de autores"
```

El archivo puede contener, por ejemplo, lo siguiente:

```
# AUTHORS

* Sergio Gómez <sergio@uco.es>
```

Hacemos push y obtenemos algo como esto:

```
$ git push
fatal: The current branch sgomez/feature-1/create-changelog has no upstream
branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin sgomez/feature-1/create-changelog
```

Como la rama es nueva, git no sabe *dónde* debe hacer push. Le indicamos que debe hacerla en *origin* y además que guarde la vinculación (equivalente al parámetro -u que vimos en el capítulo anterior). Probamos de nuevo:

```
$ git push -u origin sgomez/feature-1/create-changelog
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), 1.03 KiB | 1.03 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'sgomez/feature-1/create-changelog' on
GitHub by visiting:
remote:   https://github.com/sgomez/taller-de-git/pull/new/sgomez/feature-
1/create-changelog
remote:
To github.com:sgomez/taller-de-git.git
```

```
* [new branch]      sgomez/feature-1/create-changelog -> sgomez/feature-1/create-changelog
Branch 'sgomez/feature-1/create-changelog' set up to track remote branch 'sgomez/feature-1/create-changelog' from 'origin'.
```

Ahora la rama ya se ha subido y nos informa, además, de que podemos crear un *Pull Request* (PR). Si vamos al enlace que nos aparece veremos lo siguiente:

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master

compare: sgomez/feature-1/create-cha...

✓ Able to merge. These branches can be automatically merged.

Añadido fichero de autores

Write

Preview

AA B i “ < > ↺ ⋮ ⋮ ⋮ @ 📎 ↶

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Información

Un pull request (también conocido como "PR" en la jerga de desarrollo) es una función que facilita la colaboración en proyectos de Git, especialmente en plataformas como GitHub, GitLab o Bitbucket. Un pull request es una solicitud que un desarrollador hace para que los cambios que ha realizado en una rama de su repositorio se fusionen con otra rama, generalmente con la rama principal (como master o main).

Aquí podemos informar de en qué consiste la rama que estamos enviando. Si ya tenemos una *issue* abierta, no es necesario repetir la misma información. Podemos hacer referencia con el siguiente texto en el contenido (no en el título):

Closes #1

Esto lo que le indica a GitHub que esta PR cierra el *issues* número 1. Cuando se haga el merge de la rama, automáticamente se cerrará la incidencia.

Lo hacemos y le damos a crear.

Añadido fichero de autores #2

sgomez wants to merge 1 commit into `master` from `sgomez/feature-1/create-changeLog`

Conversation 0

Commits 1

Checks 0

Files changed 1

+4 -0

sgomez commented now

Closes #1

Añadido fichero de autores

Verified

ad22e7e

Add more commits by pushing to the `sgomez/feature-1/create-changeLog` branch on `sgomez/taller-de-git`.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request

 or view [command line instructions](#).

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Customize

Unsubscribe

Ahora podríamos hacer ya el merge con nuestra rama master, pero vamos a esperar un poco y la haremos más adelante.

Paso 2. Crear commits

A partir de ahora podemos seguir creando commits en local y enviarlos hasta que terminemos de trabajar.

Editamos el archivo AUTHORS.md .

```
# AUTHORS

* Sergio Gómez <sergio@uco.es>
* John Doe
```

Y mandamos otro commit

```
$ git commit -am "Actualizado AUTHORS.md"
$ git push
```

Si volvemos a la página de PR, veremos que aparece el nuevo commit que acabamos de enviar.

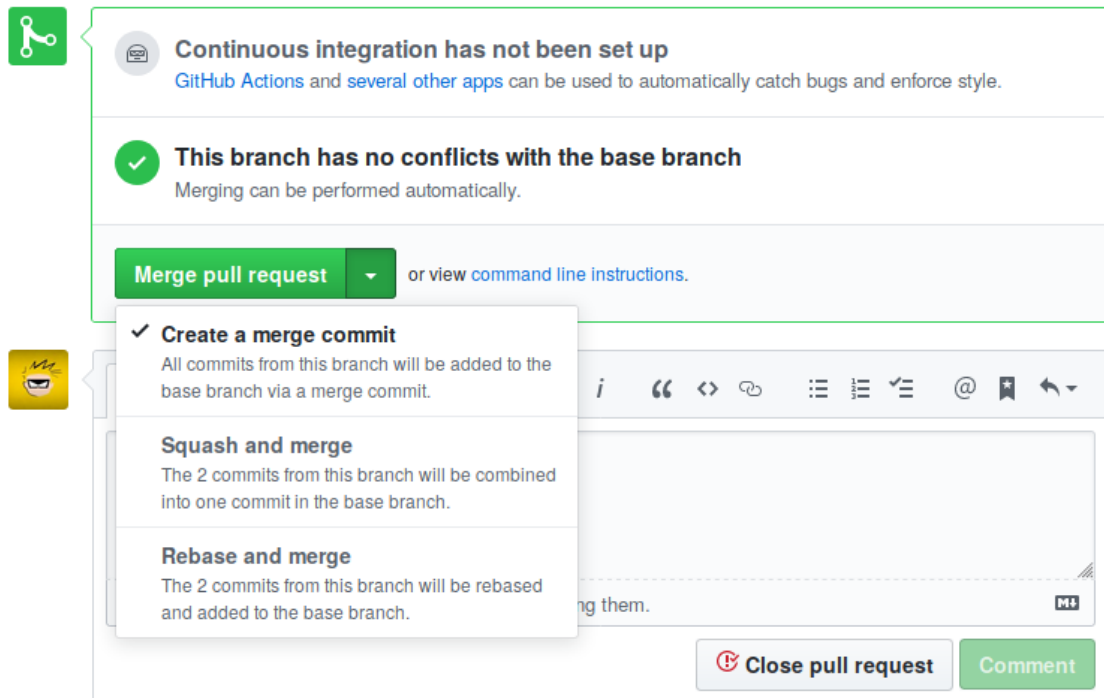
Paso 3. Discutir

GitHub permite que entre los desarrolladores se pueda abrir una discusión sobre el código, de tal manera que el trabajo de crear la rama sea colaborativo. Se puede incluso pedir revisiones

por parte de terceros y que esas revisiones sean obligatorias antes de aceptar los cambios.

Paso 4. Desplegar

Una vez que hemos terminado de crear la función de la rama ya podemos incorporar los cambios a *master*. Este trabajo ya no es necesario hacerlo en local y GitHub nos proporciona 3 maneras de hacerlo:



Crear un merge commit

Esta opción es el equivalente a hacer lo siguiente en nuestro repositorio:

```
$ git checkout main
$ git merge --no-ff sgomez/feature-1/create-changelog
$ git push
```

Es decir, el equivalente a hacer un merge entre nuestra rama y master.

Info

GitHub siempre desactiva el *fast forward*.

Crear un rebase y merge

Esta opción es el equivalente a hacer lo siguiente en nuestro repositorio

```
$ git rebase main
$ git checkout main
$ git merge --no-ff sgomez/feature-1/create-changelog
$ git push
```

Es decir, nos aseguramos de que nuestra rama está al final de *main* haciendo *rebase*, como vimos en el capítulo de ramas, y posteriormente se hace el merge.

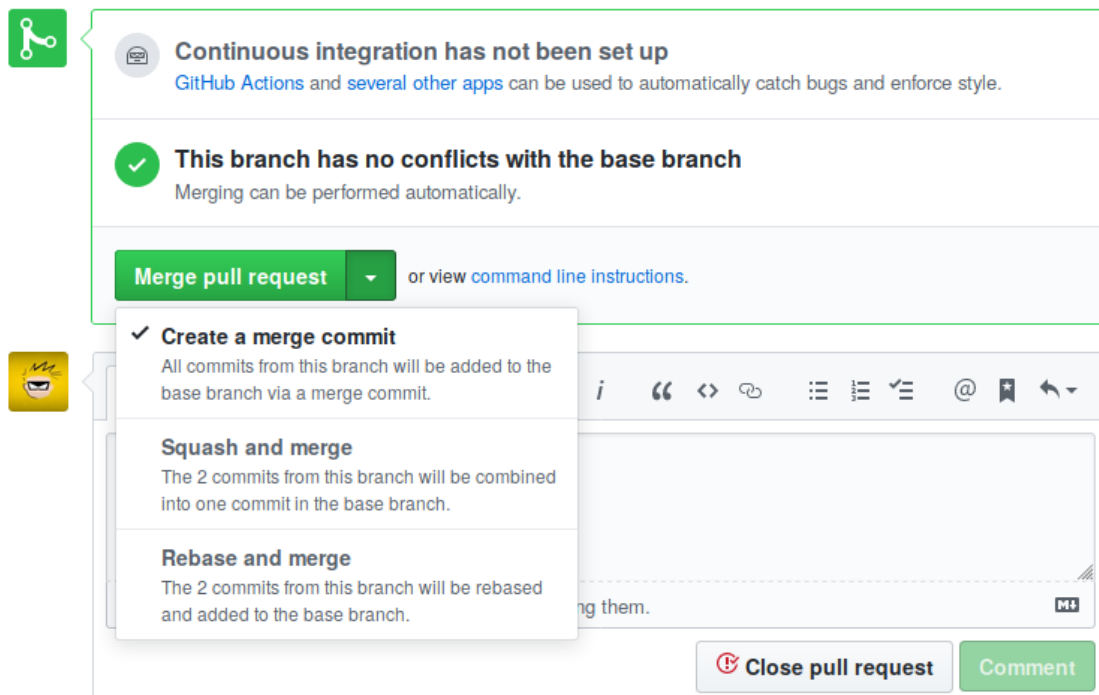
Crear un squash commit y un merge

Esta opción es el equivalente a hacer lo siguiente en nuestro repositorio:

```
$ git checkout main
$ git merge --squash sgomez/feature-1/create-changelog
$ git push
```

Esta opción es algo especial. En vez de aplicar cada uno de los commits en la rama main, ya sea directamente (*fast forward*) o no, lo que hace es crear un solo commit con los cambios de todos los commits de la rama. El efecto final es como si en la rama solo hubiera producido un solo commit.

Vamos a seleccionar este último (squash and merge) y le damos al botón para activarlo. Nos saldrá una caja para que podamos crear una descripción del commit y le damos a confirmar.



Ya hemos terminado y nos aparecerá una opción para borrar la rama, lo más recomendado para no tener ramas obsoletas.

Las consecuencias de esta acción son las siguientes:

1. El PR aparecerá como estado *merged* y en la lista de PR como cerrado.
2. El *issue* que abrimos se habrá cerrado automáticamente.
3. En el listado de commits aparecerá solo uno con un enlace al PR (en vez de los dos commits que hicimos).

Paso 5. Sincronizar

Hemos cambiado el repositorio en GitHub, pero nuestra rama master no contiene los mismos cambios que el de origen. Así que nos toca sincronizar y borrar la rama obsoleta:

```
$ git checkout main  
$ git pull --rebase --autostash  
$ git branch -D sgomez/feature-1/create-changelog
```

Info

¿Por qué *squash and merge* y no un *merge* o *rebase*? De nuevo depende de los gustos de cada equipo de desarrollo. Las características de *squash* es que elimina (relativamente) rastros de errores intermedios mientras se implementaba la rama, deja menos commits en la rama *master* y nos enlace al PR donde se implementaron los cambios.

Para algunas personas estas características son unas ventajas, para otras no. Lo mejor es experimentar cada opción y cada uno decida como quiere trabajar.