

Introducción a git



Conceptos a aprender

- Entender qué es git y cómo funciona
- Instalar y configurar git
- Registrarse en GitHub y configurar una conexión por SSH.
- Crear un repositorio en GitHub
- Crear una rama nueva
- Implementar cambios y subirlos a GitHub
- Crear una Pull Request para unificar los cambios a la rama principal.
- Resolver conflictos que puedan surgir al intentar unificar dos ramas

Introducción

En el desarrollo de software una de las competencias más necesarias es poder hacer uso de una herramienta de control de versiones. Generalmente se trabaja en equipo y es de suma importancia poder gestionar los cambios evitando conflictos y facilitando la colaboración. Git es el sistema de control de versiones más popular de la actualidad. Es gratuito y de código abierto.

Gracias a git podemos realizar un seguimiento de todas las modificaciones en el código de un proyecto. Esto nos permite regresar a versiones anteriores sin necesidad de tener guardado el proyecto en múltiples carpetas diferentes. Con git ya no es necesario tener un “proyecto final” y “proyecto final(1)” o “proyecto final(1) arreglado”. Todos los cambios son registrados en “commits” (podemos pensar en estos como ‘puntos de guardado’ o simplemente cambios), desde la creación y eliminación de archivos hasta cambios de formato y por supuesto cambios en el contenido de los mismos.

Cabe destacar que estos “commits” no los pueden ver otros usuarios hasta que hacemos un “push” y los enviamos a repositorio remoto desde donde otros colaboradores pueden hacer “pull” y obtenerlos.

Todos estos commits son registrados en un repositorio (internamente, en la carpeta oculta “.git”) que puede ser local como remoto. Si bien git es un sistema distribuido, todo el funcionamiento del control de versiones, incluida la historia está disponible en las máquinas locales, generalmente se hace uso de un sitio remoto desde donde se centraliza el código y desde donde cada colaborador interactúa con el código.

TODO

¿Es necesario saber usar git?

Git es actualmente considerado un “commodity”. El manejo de git puede parecer un poco intimidante al comienzo, especialmente si se tiene poca experiencia con manejo de consola, pero con práctica y esfuerzo podrás dominarlo.

Flujo básico de trabajo con git

- Crear un repositorio, generalmente con un servicio como GitHub.
- Clonar el repositorio en local.
- Crear una branch (rama) nueva.
- Aplicar cambios al repositorio local
- Hacer un “commit” (confirmación) de los cambios realizados.
- Hacer un “push” (envío) de los cambios al repositorio remoto.
- En la herramienta online hacemos una “pull request” o “PR” (petición de aplicar cambios a la branch principal).
- Revisar los cambios a aplicar en la PR.
- Hacer un merge (fusión) a la branch principal.

¿Git o Github?

Git no depende necesariamente de GitHub. GitHub es una plataforma de hospedaje de código que utiliza git y lo expande con funcionalidades gráficas que ayudan al trabajo colaborativo. Por otro lado, git es la herramienta de control de software en sí. Dos desarrolladores podrían usar git sin necesidad de GitHub o similar conectando sus repositorios usando SSH o similar (estableciendo como ‘remoto’ a la máquina del otro colaborador), pero es mucho más sencillo y fácil de controlar hacer uso de un servicio como GitHub.

Instalar y configurar Git en Ubuntu

Instalación

Antes que nada, deberíamos verificar si git ya está instalado en nuestra maquina local. Para ello, ejecutamos el siguiente comando:

```
git --version
```

Si la consola devuelve un mensaje informando la versión de git instalada (por ejemplo 2.34.1), significa que podemos saltarnos la instalación e ir directamente a la configuración.

Si git no está instalado en nuestro local, la forma más sencilla de instalarlo es usando los paquetes oficiales de Ubuntu. Para eso hacemos uso del manejador de paquetes de Ubuntu APT.

Primero nos aseguramos de que el índice de paquetes esté actualizado usando apt update:

```
sudo apt update
```

Una vez que la actualización se termine, pasamos a instalar git:

```
sudo apt install git
```

Al terminar, podemos verificar que Git está efectivamente instalado corriendo el comando:

```
git --version
```

Si la consola nos devuelve un número de versión, significa que git se instaló correctamente.

Configuración

Para configurar git hacemos uso del comando git config. Necesitamos especificar nuestro nombre e e-mail ya que git embebe esta información en cada commit que hagamos. Podemos lograr esto ejecutando los siguientes comandos:

```
git config --global user.name "Tu Nombre"  
git config --global user.email "tunombre@mail.com"
```

Para verificar que los datos fueron cargados correctamente, podemos hacer uso del siguiente comando:

```
git config --list
```

Crear una cuenta de GitHub

Para poder trabajar con GitHub, primero necesitaremos registrarnos en la plataforma. Para hacerlo, vamos a la sección "Pricing" del sitio: <https://github.com/pricing>

Dentro de esa sección, hacemos clic en el botón "Join for free".



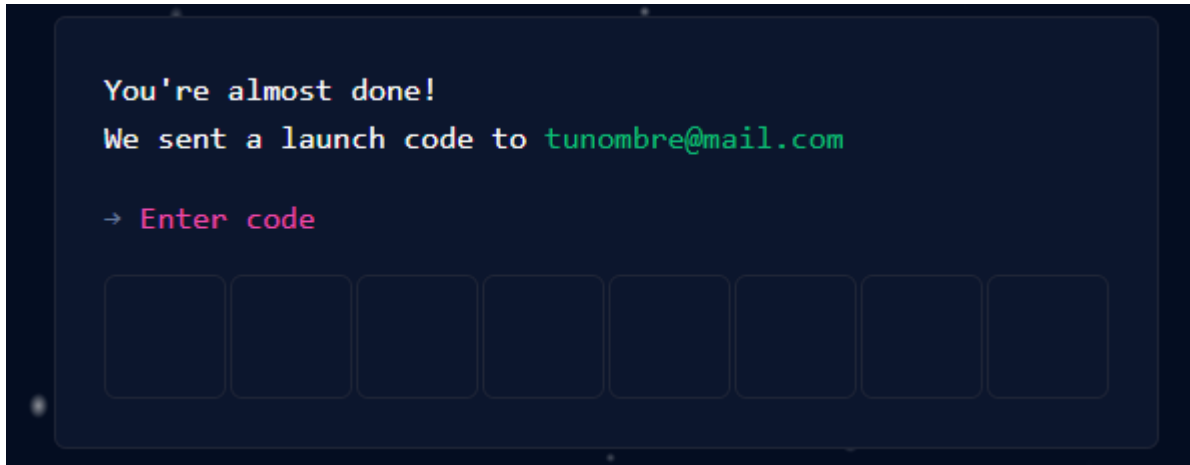
Join for free

Esto nos llevará a una sección donde tenemos que definir nuestro nombre de usuario, e-mail y clave. Es importante que el e-mail que usemos sea uno al que tengamos acceso.

Luego de insertar todos los datos necesarios, debajo de la sección "Verify your account" hacemos la validación de que somos humanos para finalmente hacer clic en "Create account"

Create account

Nos llegará un e-mail con el código de verificación (Con asunto “🚀 Your GitHub launch code”)



Ingresamos el código y por ahora podemos hacer clic en el link de “Skip personalization” para poder comenzar a trabajar con nuestra cuenta.

Configurar SSH de Github

Usando el protocolo SSH, podemos, de manera segura, conectarnos y autenticarnos a GitHub sin necesidad de proveer datos de usuario con cada conexión.

Generar una llave SSH

Abrimos la terminal y ejecutamos el comando:

```
ssh-keygen -t ed25519 -C "tumail@ejemplo.com"
```

(Reemplazando el e-mail por el que usamos con nuestra cuenta de GitHub).

Luego de generar la clave, necesitamos agregarla al ssh-agent.

Ejecutamos ssh-agent en segundo plano:

```
eval "$(ssh-agent -s)"
```

Ejecutamos el comando para agregar la llave:

```
ssh-add ~/.ssh/id_ed25519
```

(Si usaste un nombre distinto para la llave, hay que reemplazar id_ed25519 por el nombre correcto)

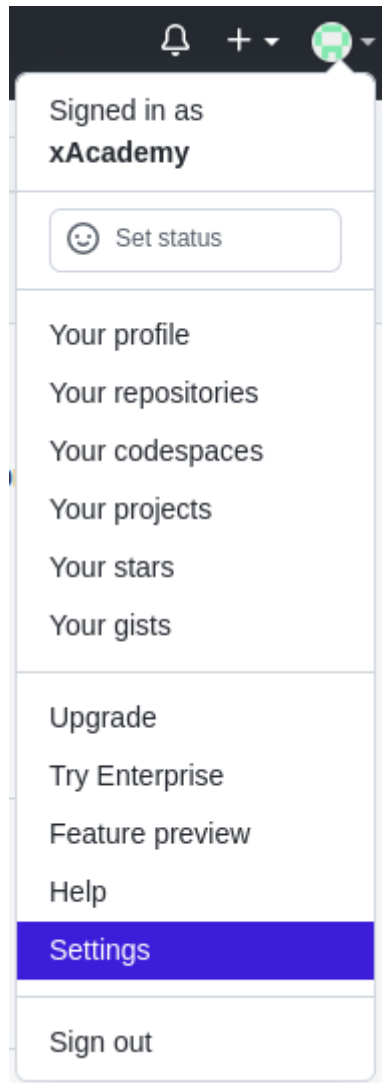
Agregar la llave SSH a la cuenta de Github

Usando el comando cat, mostramos el contenido de la llave pública.

```
cat ~/.ssh/id_ed25519.pub
```

Luego, seleccionamos la llave pública y lo copiamos al portapapeles.

En GitHub, vamos a la parte superior derecha y hacemos clic en nuestro perfil para luego expandir el menú y hacemos clic en “Settings”



En la pantalla de configuración del perfil, en el sidebar a la izquierda, bajo la sección “Access”, podemos encontrar la opción “SSH and GPG keys”.

Dentro de la sección “SSH and GPG keys” hacemos clic en el botón “New SSH Key”

New SSH key

En el campo “Title” agregamos una descripción de la llave. Por ejemplo, “PC Personal”. Esto es importante ya que si en el futuro perdemos acceso a la computadora que tiene esta llave, podremos fácilmente encontrar la llave y desactivarla.

Dejamos el campo “Key type” con el valor por defecto, “Authentication Key”.

En el campo “Key” pegamos el contenido de nuestra llave pública.

Finalmente, hacemos clic en el botón “Add SSH key”.

Add SSH key

Validar la conexión SSH

En la terminal ejecutamos el comando

```
ssh -T git@github.com
```

Seguramente primero responderá con un mensaje como este:

```
> The authenticity of host 'github.com' can't be established.  
> RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.  
> Are you sure you want to continue connecting (yes/no)?
```

Respondemos con “yes” y luego obtendremos un mensaje como este:

```
Hi xAcademy! You've successfully authenticated, but GitHub does not provide  
shell access.
```

Si no encuentra el host

En caso de que al intentar validar la conexión SSH obtengamos como respuesta un mensaje como este:

```
ssh: connect to host github.com port 22: No route to host
```

Debemos hacer unos cambios en el archivo de configuración de ssh (o crearlo, en el caso de que no exista o esté vacío). En este caso usamos vim, pero podríamos usar nano u otra herramienta de edición de texto:

```
sudo vim ~/.ssh/config
```

Esto abrirá el archivo de configuración de ssh. Debemos entrar en el modo de inserción pulsando la tecla “i” y luego insertar lo siguiente:

```
Host github.com  
  Hostname ssh.github.com  
  Port 443
```

Luego pulsamos “esq” para salir del modo de inserción y pulsamos “:wq” para salir y guardar el archivo.

Si volvemos a probar la conexión por ssh con github, deberíamos tener acceso sin problemas.

```
ssh -T git@github.com
> The authenticity of host 'github.com' can't be established.
> RSA key fingerprint is SHA256:nThbg6kXUpJWGL7E1IGOCspRomTxdCARLviKw6E5SY8.
> Are you sure you want to continue connecting (yes/no)? yes
...
Hi xAcademy! You've successfully authenticated, but GitHub does not provide
shell access.
```

Crear un repositorio

El repositorio es donde guardamos todo lo que necesita nuestro proyecto. Tenemos dos maneras de inicializar un repositorio, primero está la opción de hacer localmente y segundo desde GitHub.

Crear repositorio local

Primero nos dirigimos al directorio local donde queremos que esté el repositorio:

```
cd /directorio
```

Luego, para convertir el directorio en repositorio git usamos el comando git init

```
git init
```

Cabe destacar que la branch principal generalmente es “master”, pero por convención actualmente se considera que “main” debería ser la branch principal. Si deseamos cambiarla, tenemos que cambiar la configuración global de nuestro git:

```
git config --global init.defaultBranch main
```

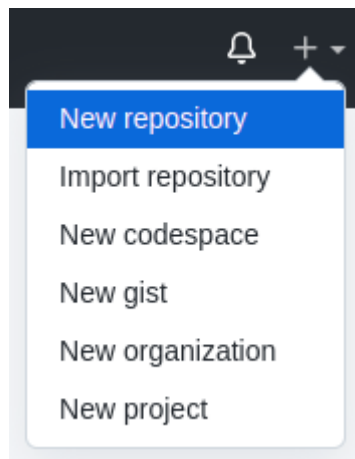
Ya creado el repositorio, la carpeta oculta “.git” contiene toda la información que git necesita para funcionar. Si queremos ver esta carpeta tenemos que usar el siguiente comando:

```
ls -la
```

Crear repositorio en GitHub

La segunda manera y tal vez la más habitual en el trabajo colaborativo, es crear el repositorio desde GitHub.

Primero vamos a hacer clic en el “+” en la parte superior derecha de la pantalla y luego, en el menú que se abre, hacemos clic en “New repository”



En el formulario que se abre, necesitamos definir un nombre de repositorio, si será público o privado (generalmente para hacer pruebas, es recomendable usar repositorios privados), si necesitamos un archivo README, configuración base de .gitignore y el tipo de licencia. En nuestro caso, solo seleccionaremos “Private” y “Add a README file”.



Owner *

Repository name *

 xAcademy ▼ / ejemplogit ✓

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

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](#).

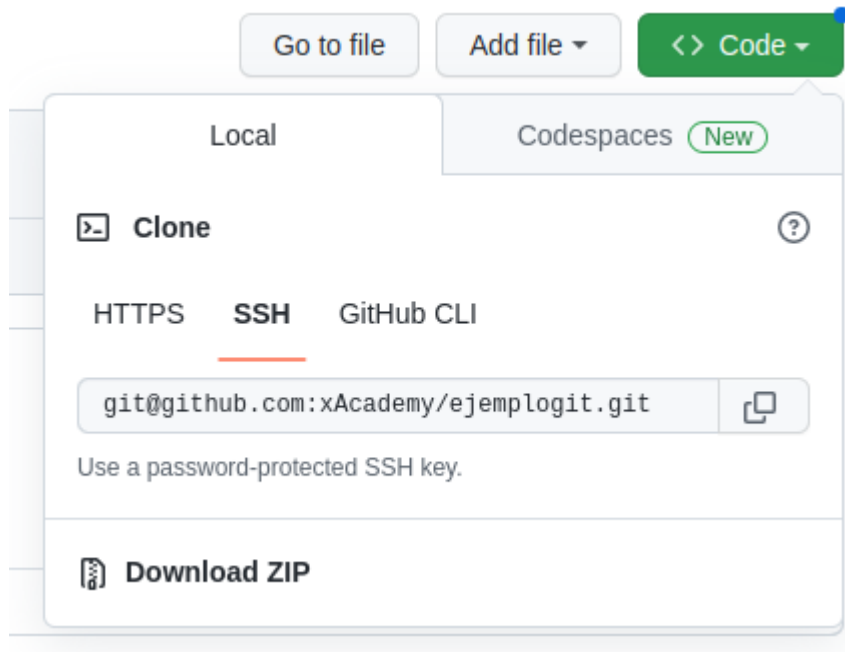
 You are creating a private repository in your personal account.

Create repository

Seremos redireccionados a la URL de nuestro nuevo repositorio. El único archivo presente por ahora será el “README.md”, el cual podremos editar directamente desde git. Usa la sintaxis md.

Clonar repositorio

Para utilizar el repositorio que creamos en GitHub, debemos clonarlo en nuestro local. Para obtener la ruta correcta del repositorio remoto, vamos a la pantalla de inicio del repositorio en GitHub. Haciendo clic en el botón verde “Code” se expande un menú de clonación:



Nos interesa la pestaña que dice “SSH”. Copiamos al portapapeles la ruta.

En nuestra consola, ejecutamos el comando git clone:

```
git clone git@github.com:xAcademy/ejemplologit.git
Cloning into 'ejemplologit'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
Receiving objects: 100% (3/3), done.
```

Git se encargará de crear la carpeta del repositorio (en este caso “ejemplologit”).

Crear una branch

Los repositorios en git usan ramas o “branches” para organizar diferentes versiones del código. Generalmente cada nueva funcionalidad se trabaja en una branch nueva basada en la principal. Una vez que la nueva funcionalidad está codeada y probada, se la puede integrar a la branch principal.

Crear branch localmente

Hay varias maneras de crear una branch desde la consola, la primera forma es usando git branch


```
git branch ejemplo-branch
```

Esto creará la branch “ejemplo-branch”, podemos verificar que efectivamente se creó usando git branch

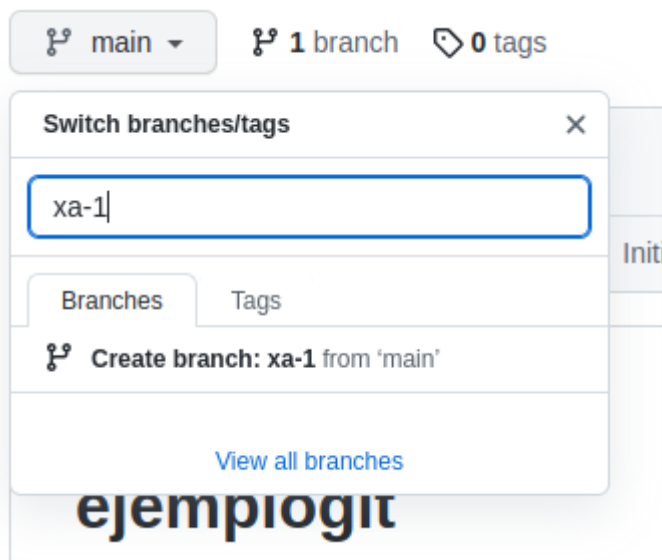
```
git branch
ejemplo-branch
* main
```

Cabe destacar que esta manera de crear una branch NO mueve automáticamente la branch seleccionada a la que acaba de ser creada. Para hacer esto, debemos hacer git checkout, que veremos un poco más adelante.

Crear branch en GitHub


En GitHub, para crear una branch nueva, hacemos clic en el botón de lista desplegable que tiene el icono . Generalmente al inicio tiene el nombre de la branch principal que es “main” por defecto.

Se abre un formulario de dialogo y escribimos el nombre de la branch que queremos crear (este menú también sirve para buscar branches que ya existen)

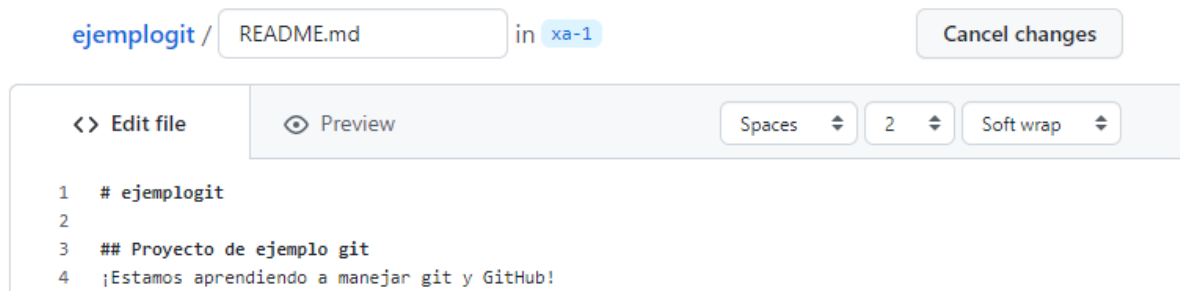


En nuestro caso, el nombre de la nueva branch es “xa-1”. Es buena idea identificar de forma unívoca una branch para relacionarlo a alguna tarea o funcionalidad. Luego de escribir el nombre que queremos para la branch, hacemos clic en “Create branch”. Es buena idea prestar atención a la branch desde la que se crea la nueva, en este caso es “main”, es decir, la nueva branch xa-1 será una copia de main hasta que se inserten nuevos cambios.

Hacer cambios desde GitHub

En la nueva branch “xa-1” vamos a hacer clic en el archivo README.md y hacer clic en el botón de edición .

En el editor que aparece, podemos escribir cualquier cosa, como éste es el archivo “README” lo lógico sería escribir algo sobre el proyecto en sí.



Luego de terminar de escribir nuestros cambios, bajamos a la parte inferior de la pantalla. Ahí nos encontraremos con un dialogo como este:

Commit changes

Update README.md

Add an optional extended description...

☒ Commit directly to the `xa-1` branch.
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

Cancel

Aquí es donde escribiremos el mensaje de nuestro commit. Cada equipo de trabajo tiene sus propias normas sobre cómo deben ser los mensajes, pero mientras más nos explayemos y hagamos referencia al requerimiento, mejor ya que esto provee información invaluable a futuro. Luego de completar nuestro mensaje, hacemos clic en “commit changes” para confirmar los cambios.

Obtener cambios en nuestro local

En nuestro ambiente local, si revisamos la lista de branches/ramas disponibles, notaremos que aún no está la nueva branch “xa-1”.

```
git branch
ejemplo-branch
* main
```

Incluso si ejecutamos el comando `git branch -r` para ver la lista de branches remotos, no veremos la nueva branch “xa-1”:

```
git branch -r
origin/HEAD -> origin/main
origin/main
```

Necesitamos ejecutar dos comandos, primero obtener los cambios del repositorio remoto y segundo cambiar la branch activa de local a la nueva branch que acabamos de obtener

Obtener cambios remotos

Para actualizar nuestro repositorio local con los datos remotos, hacemos uso del comando git fetch:

```
git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
Unpacking objects: 100% (3/3), 697 bytes | 697.00 KiB/s, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
From github.com:xacademy/ejemplogit
* [new branch]      xa-1      -> origin/xa-1
```

Revisando de nuevo el listado de branches, esta vez usando la opción “-a” para ver branches locales y remotas, podremos notar que efectivamente está la branch remota “xa-1”:

```
git branch -a
ejemplo-branch
* main
remotes/origin/HEAD -> origin/main
remotes/origin/main
remotes/origin/xa-1
```

Mover la branch actual a la nueva

Para movernos a la branch “xa-1” usamos git checkout xa-1:

```
git checkout xa-1
Branch 'xa-1' set up to track remote branch 'xa-1' from 'origin'.
Switched to a new branch 'xa-1'
```

Podemos notar que git nos informa que la branch local ‘xa-1’ está conectada con la branch remota ‘xa-1’.

Podemos revisar el estado de la branch local usando el comando git status:

```
git status
On branch xa-1
Your branch is up to date with 'origin/xa-1'.
```

```
nothing to commit, working tree clean
```

Hacer un commit desde nuestro local

En nuestro ambiente local, modificamos nuevamente el archivo README.md. Podemos usar cualquier editor de texto, desde vim, hasta vscode.

```
# ejemplomit

## Proyecto de ejemplo de git
¡Estamos aprendiendo a manejar git y GitHub!

### Probando un cambio desde local
Vamos editar este archivo y subirlo como un commit nuevo.
```

Luego de guardar el archivo, si volvemos a usar git status, nos encontraremos con una respuesta diferente:

```
git status
On branch xa-1
Your branch is up to date with 'origin/xa-1'.

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

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

Agregar archivos a la lista de cambios

Para poder enviar los cambios que hicimos al archivo, debemos primero agregarlos a la lista de cambios. Esto lo logramos con el comando 'git add' y si volvemos a comprobar el estado, notaremos una diferencia:

```
git add README.md
git status
On branch xa-1
Your branch is up to date with 'origin/xa-1'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

Crear el commit y enviarlo al remoto

Lo siguiente a hacer es agregar un nuevo commit para 'confirmar' los cambios.

```
git commit -m "Modificando README.md localmente"
[xa-1 36d33aa] Modificando README.md localmente
1 file changed, 3 insertions(+)
```

Podemos verificar que el commit se creó usando git log:

```
git log --graph --pretty=format:'%h - %s (%cr) <%an>' --abbrev-commit
* 36d33aa - Modificando README.md localmente (1 minute ago) <xacademy>
* 7966e05 - Update README.md (5 minutes ago) <xacademy>
* 50deab4 - Initial commit (10 minutes ago) <xacademy>
```

Ahora que tenemos listo nuestro commit, podemos enviarlo al repositorio remoto en GitHub.

```
git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 407 bytes | 407.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:xacademy/ejemplogit.git
7966e05..36d33aa xa-1 -> xa-1
```

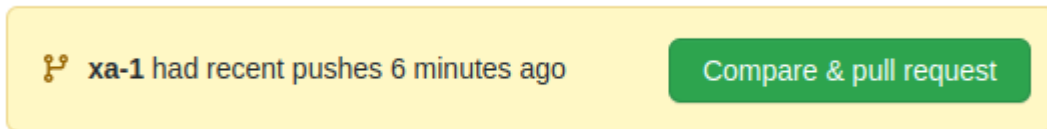
Si volvemos a GitHub, a la branch xa-1, nos encontraremos con el archivo README.md modificado.



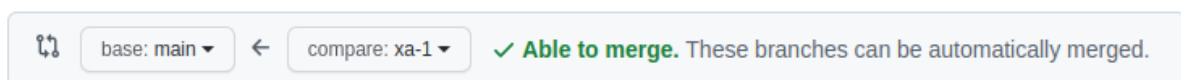
Hacer una Pull Request en GitHub

Ahora que nuestros cambios están listos, podemos enviarlos a la branch principal de nuestro proyecto. Si bien podríamos hacerlo en nuestro ambiente local, generalmente es buena práctica proteger las branches principales para evitar los cambios sin verificación por un par. Para esto, existen las Pull Request.

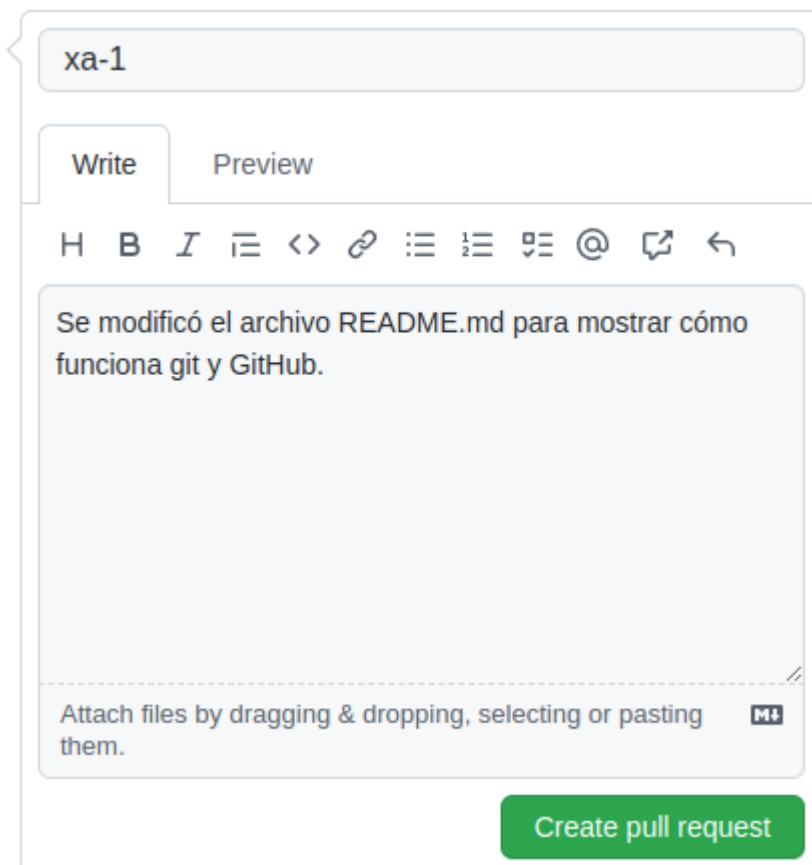
En GitHub aparece un mensaje cuando recientemente una branch recibió modificaciones:



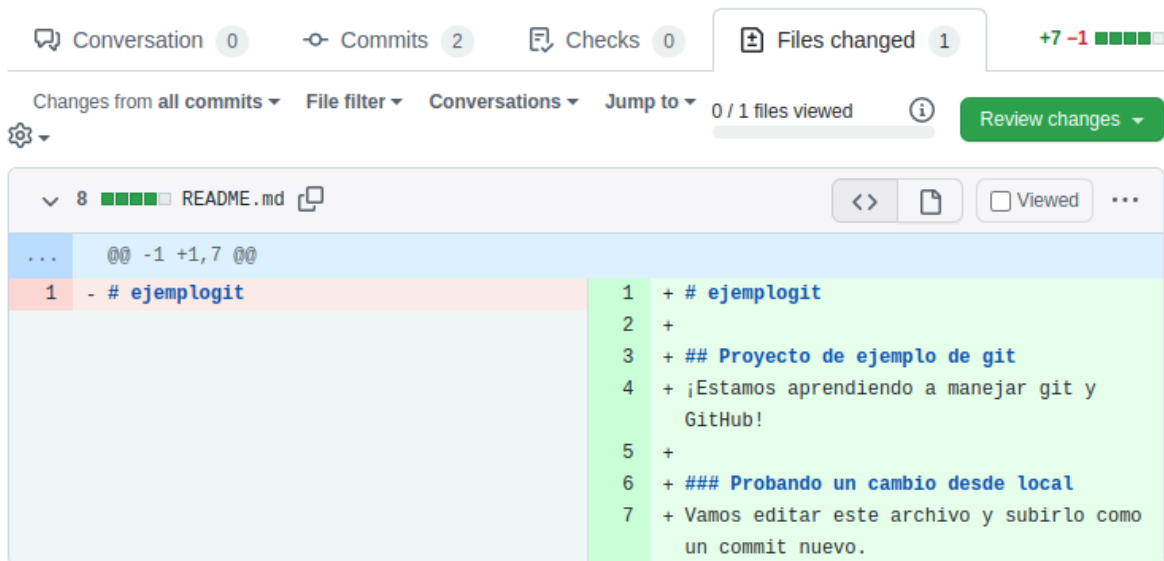
Al hacer clic en “Compare & pull request” seremos redirigidos al formulario para crear una nueva Pull Request. Por defecto, la branch a la que se hace el merge es “main”:



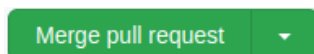
Si vemos el mensaje “Able to merge”, significa que no hay conflictos y podemos avanzar sin problemas. Escribimos un título para la PR y un mensaje:



Luego de hacer clic en “Create pull request” se creará efectivamente la PR. En la pestaña “Files changed” se puede ver las modificaciones.

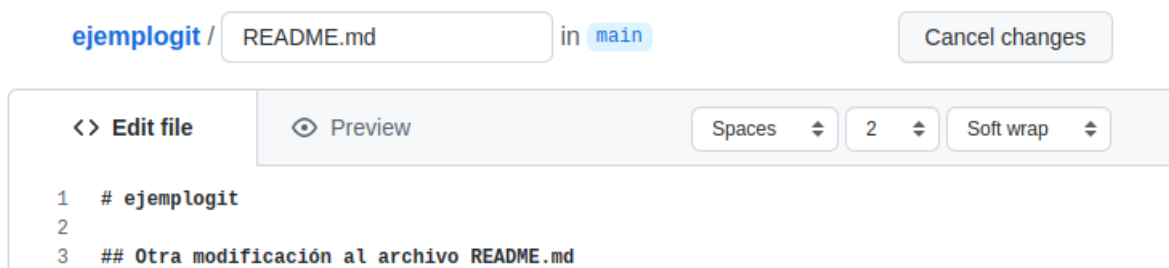


Otro usuario podrá revisar nuestros cambios y aplicarlos para unificarlos a la branch main haciendo clic en el botón “Merge pull request”




Resolver conflictos

Puede que nuestra Pull Request no resulte en el ‘camino feliz’ y encontremos que algún archivo tiene ‘conflictos’ con la branch en la que queremos hacer el merge. Digamos que otro compañero editó el archivo README.md en la branch main antes de que se completara la PR:



Esto generará que en la PR, el botón de “Merge pull request” quede grisado. Si los conflictos son simples, podremos resolverlos directamente en GitHub, pero si son demasiado complejos, será necesario hacerlo en nuestro ambiente local.



This branch has conflicts that must be resolved

Use the [web editor](#) or the [command line](#) to resolve conflicts.

Conflicting files

README.md


Merge pull request ▼

or view [command line instructions](#).

Al hacer clic en el botón “Resolve conflicts” se abre el editor web desde donde podemos resolver los conflictos.

xa-1 #1

Resolving conflicts between `xa-1` and `main` and committing changes → `xa-1`

1 conflicting file	README.md 1 conflict Prev ^ Next v ⚙️ Mark as resolved
 READM... README.md	<pre>1 # ejempligit 2 3 <<<<<< xa-1 4 ## Proyecto de ejemplo de git 5 ¡Estamos aprendiendo a manejar git y GitHub! 6 7 ### Probando un cambio desde local 8 Vamos editar este archivo y subirlo como un commit nuevo. 9 ===== 10 ## Otra modificación al archivo README.md 11 >>>>>> main</pre>

Como en este caso es un archivo de texto, es simple darse cuenta de que podemos incluir ambas modificaciones sin romper nada. En el caso de resolver conflictos de código, requiere mucha atención ya que un desarrollador puede llamar a un método de una forma y otro de otra distinta.

xa-1 #1

Resolving conflicts between xa-1 and main and committing changes → xa-1

1 conflicting file

 READM...
README.md

README.md 1 conflict Prev ^ Next v ⚙

Mark as resolved

1 # ejemplomit

2

3 ## Proyecto de ejemplo de git

4 ¡Estamos aprendiendo a manejar git y GitHub!

5

6 ### Probando un cambio desde local

7 Vamos editar este archivo y subirlo como un commit nuevo.

8

9 ## Otra modificación al archivo README.md

Una vez completadas las correcciones, podemos hacer clic en “Mark as resolved” y luego en “commit merge”. Esto agregará un nuevo commit que nos permitirá completar el merge de la PR.

Próximos pasos

Felicidades, ya tenés un concepto básico de cómo manejar git y GitHub. Pero dominar a git requiere de mucha investigación y práctica. Los siguientes son algunos conceptos que recomendamos investigar para tener un entendimiento más redondo de todo lo que se puede hacer con git:

- El archivo .gitignore
- Uso del stash
- Uso de cherry pick
- Flujos de trabajo
- Git Hooks.
- Automatización de builds usando git