

Módulo Herramientas de Desarrollo

git

Objetivos: Descubrir el control de versionado usando git y sus comandos básicos. Poder conectarse a GitHub y realizar actividades esenciales del desarrollo de software.

Introducción

En el desarrollo de software, es común trabajar en equipo en el mismo proyecto. A medida que diferentes personas trabajan en el código, es importante asegurarse de que las diferentes versiones no entren en conflicto. Git es una herramienta que permite controlar las diferentes versiones de un proyecto de software y colaborar de manera más eficiente. Es como un sistema de guardado de videojuegos que permite volver a un punto anterior si algo sale mal.

Git te permite guardar todas las modificaciones que hagas en un archivo de código (cambios como agregar o eliminar una línea de código) en "commits", como puntos de guardado. Si algo sale mal, puedes volver a un "commit" anterior en lugar de tener que comenzar desde cero. También puedes trabajar en diferentes versiones del proyecto al mismo tiempo, y git te ayuda a combinarlas sin conflictos.

Git funciona en tu propia computadora, pero también puede funcionar en un servidor en línea llamado repositorio. Si trabajas en equipo, puedes subir tus cambios al repositorio para que otros miembros del equipo puedan verlos y trabajar en ellos. Git también te permite comparar diferentes versiones del proyecto y ver exactamente qué ha cambiado.

En resumen, git es una herramienta que te ayuda a trabajar en equipo y controlar las diferentes versiones de un proyecto de software. Te permite guardar todos los cambios, trabajar en diferentes versiones del proyecto y compartir tus cambios con otros miembros del equipo.

¿Es necesario saber usar git?

Como desarrollador de software que trabaja en equipo, el conocimiento y uso de git es muy valioso. Git te permite colaborar con otros desarrolladores de manera eficiente y mantener un

registro de todos los cambios realizados en el código. Aunque puede parecer un poco abrumador al principio, con práctica y esfuerzo podrás dominarlo.

No te preocupes si no tienes experiencia previa con la línea de comandos o el manejo de git. Hay muchas herramientas visuales que puedes utilizar para hacer que el proceso de git sea más fácil. Además, hay muchos tutoriales en línea y recursos disponibles que pueden ayudarte a aprender git a tu propio ritmo.

En resumen, si quieres ser un desarrollador de software en equipos, es muy recomendable que aprendas a utilizar git. Puede parecer intimidante al principio, pero con práctica y esfuerzo podrás dominarlo. Hay muchas herramientas y recursos disponibles para ayudarte a aprender git a tu propio ritmo, así que no te preocupes si no tienes experiencia previa.

Flujo básico de trabajo con git

1. **Crear un repositorio:** Usando un servicio en línea como GitHub, crea un nuevo repositorio para tu proyecto.
2. **Clonar el repositorio en local:** Descarga una copia del repositorio en tu computadora para que puedas trabajar en él.
3. **Crear una nueva rama:** Crea una nueva rama para que puedas trabajar en cambios sin afectar la rama principal del proyecto.
4. **Realizar cambios:** Realiza cambios en los archivos de tu proyecto en tu computadora.
5. Hacer un "commit": Guarda tus cambios en un "commit" para que puedas hacer un seguimiento de los cambios realizados.
6. **Hacer un "push":** Envía tus cambios al repositorio en línea para que otros miembros del equipo puedan verlos.
7. **Hacer una "pull request" o "PR":** Solicita que los cambios realizados en tu rama se fusionen con la rama principal del proyecto.
8. **Revisar los cambios en la PR:** Otros miembros del equipo pueden revisar tus cambios y hacer comentarios.
9. **Hacer un merge:** Si tus cambios son aprobados, realiza una fusión con la rama principal del proyecto.

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
```

Configurar SSH de Github

Al configurar una conexión SSH con GitHub, podemos conectarnos y autenticarnos de manera segura sin tener que proporcionar nuestros datos de usuario cada vez que nos conectamos.

Esto nos permite trabajar de manera más eficiente con repositorios remotos en GitHub y asegura la privacidad de nuestras credenciales de usuario. La configuración de SSH también permite una conexión cifrada y más segura entre nuestro equipo local y el servidor de GitHub. La configuración de SSH es una práctica recomendada para cualquier desarrollador que utilice GitHub para alojar proyectos de software.

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

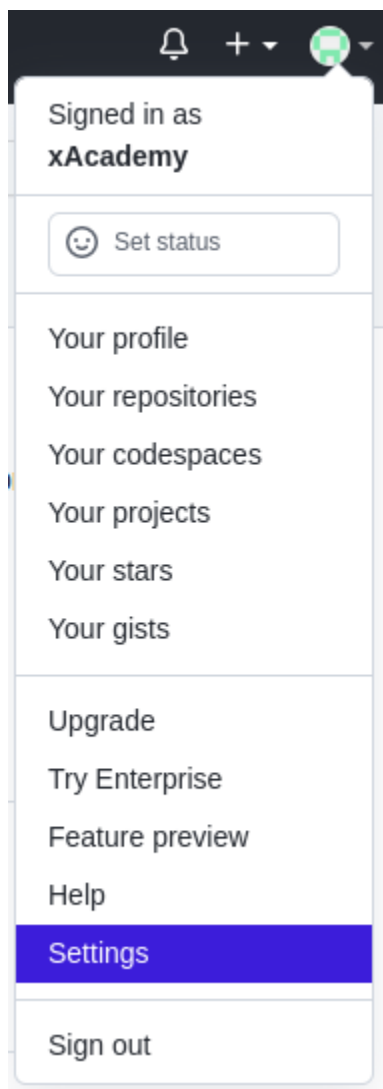
Primero, vamos a la terminal de línea de comandos y utilizamos el comando "cat" para mostrar el contenido de nuestra llave pública. El comando que debemos utilizar es:

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

A continuación, seleccionamos todo el contenido de la llave pública que se muestra en la terminal y lo copiamos al portapapeles.

Luego, nos dirigimos a la página de configuración de nuestra cuenta en GitHub. Para hacer esto, hacemos clic en nuestro perfil en la parte superior derecha de la página de GitHub y luego seleccionamos "Settings" en el menú desplegable.

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 sección "Access" del sidebar, seleccionamos "SSH and GPG keys".

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

New SSH key

Agregamos una descripción a nuestra llave, como "Mi PC personal".

En el campo "Key", pegamos el contenido de nuestra llave pública que obtuvimos previamente.

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:nThbg6kXUPJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
```

```
> 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

Un repositorio es donde guardamos todos los archivos y carpetas necesarios para nuestro proyecto de software. Existen dos maneras de inicializar un repositorio: podemos crear uno localmente en nuestra máquina o podemos crear uno en GitHub.

Crear repositorio local

Para crear un repositorio localmente, utilizamos el comando "git init" en la terminal desde la carpeta donde queremos almacenar nuestro proyecto. Esto crea un repositorio local en nuestra máquina.

```
git init
```

Es importante destacar que la rama principal en un repositorio de Git generalmente se llama "master". Sin embargo, actualmente se considera que "main" debería ser la rama principal en su lugar. Si deseamos cambiar la rama principal de nuestro repositorio, podemos hacerlo cambiando la configuración global de Git.

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

Una vez que creamos un repositorio de git en nuestra máquina, se crea una carpeta oculta llamada ".git" que contiene toda la información necesaria para que git funcione. Si queremos ver el contenido de esta carpeta, podemos utilizar el siguiente comando en la terminal:

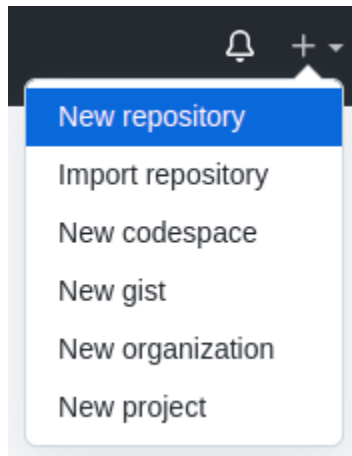
```
ls -a
```

Este comando lista todos los archivos y carpetas, incluyendo los ocultos, en el directorio actual. De esta manera, podemos ver y acceder a la carpeta ".git" y a todos sus archivos y subdirectorios para comprender mejor cómo Git está registrando y controlando los cambios en nuestro proyecto. Es importante tener en cuenta que, en general, no necesitamos modificar directamente la carpeta ".git", ya que Git se encarga de manejar todo de forma automática.

Crear repositorio en GitHub

La forma más común de crear un repositorio es hacerlo desde GitHub, especialmente cuando trabajamos en colaboración con otros desarrolladores.

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



Se abrirá un formulario que nos permitirá definir la configuración básica de nuestro repositorio. En este formulario, necesitamos proporcionar un nombre para el repositorio y elegir si deseamos que sea público o privado. Es recomendable que para las pruebas iniciales de nuestro proyecto, utilicemos un repositorio privado.

También podemos elegir si deseamos agregar un archivo README.md, que es un archivo de texto que sirve como introducción y descripción general de nuestro proyecto. Además, podemos elegir el tipo de licencia para nuestro proyecto, lo que puede ser útil si planeamos compartir nuestro código con otros desarrolladores.

En nuestro caso, seleccionaremos "Private" y "Add a README file" en el formulario. Una vez que hayamos proporcionado toda la información necesaria, hacemos clic en el botón "Create repository" para crear nuestro nuevo repositorio. Luego, podemos comenzar a agregar archivos y colaborar con otros desarrolladores en nuestro proyecto.

Owner *



xAcademy ▾

Repository name *

/ 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.

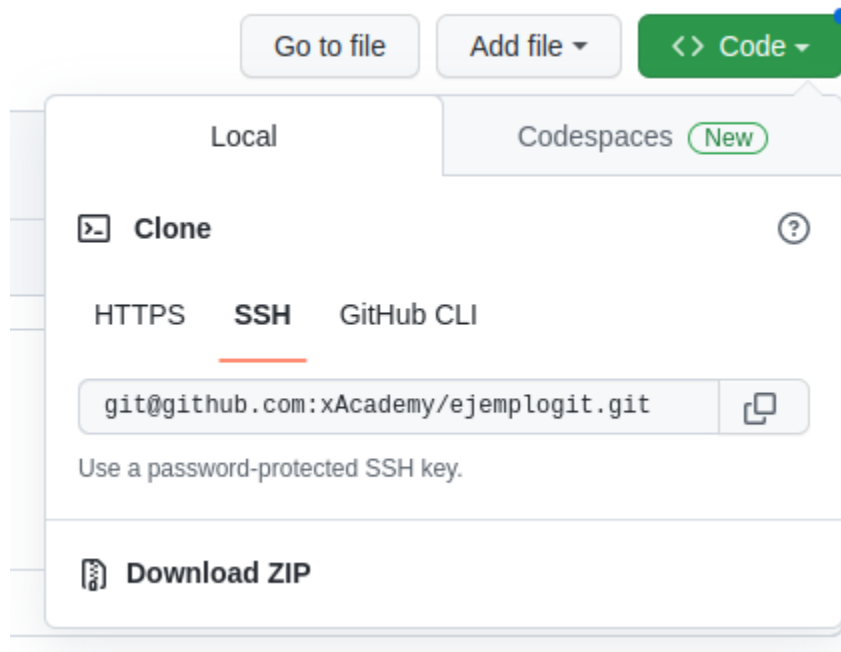
Clonar repositorio



Technology
withPurpose
Foundation

santex

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/ejemplogit.git
Cloning into 'ejemplogit'...
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 “ejemplogit”).

Crear una branch

En git, los repositorios utilizan ramas o "branches" para organizar diferentes versiones del código. Por lo general, se trabaja en una nueva funcionalidad en una rama separada, que se basa en la rama principal del repositorio. Una vez que se ha completado la nueva funcionalidad y se han realizado las pruebas necesarias, se puede integrar en la rama principal.

Cada rama es como una línea de tiempo separada del proyecto, lo que permite trabajar en diferentes funcionalidades y correcciones de errores de forma aislada. Esto hace que sea más

fácil revertir los cambios si algo no funciona correctamente o si se necesita volver a una versión anterior del código. En resumen, el uso de ramas en git facilita la gestión de diferentes versiones del código y permite a los desarrolladores trabajar de forma más eficiente y colaborativa.

Crear branch localmente

Existen varias formas de crear una nueva rama en git desde la consola. Una de ellas es usando el comando `git branch`, seguido del nombre que deseamos darle a la nueva rama. Por ejemplo, si queremos crear una rama llamada "ejemplo-branch", podemos ingresar lo siguiente en la consola:


```
git branch ejemplo-branch
```

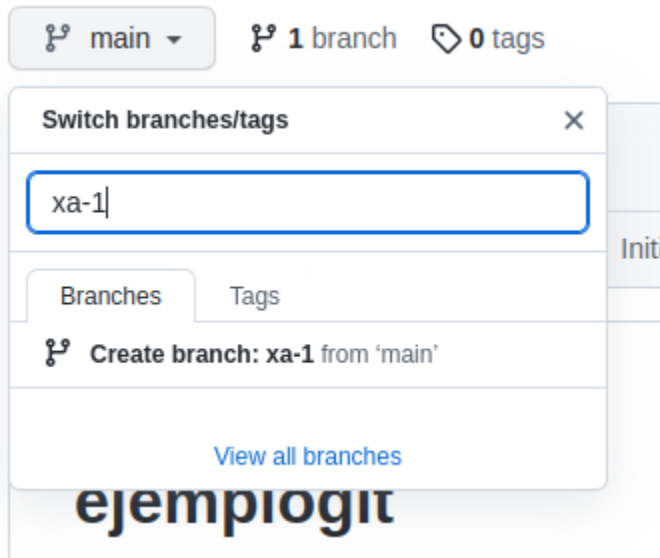
Esto creará una nueva rama en el repositorio con el nombre "ejemplo-branch". Para verificar que la nueva rama ha sido creada, podemos usar el comando `git branch` de nuevo:

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

Como podemos ver, la nueva rama se ha agregado a la lista de ramas disponibles, pero el asterisco indica que todavía estamos en la rama "main". Es importante tener en cuenta que crear una nueva rama no mueve automáticamente la rama actual a la nueva rama. Para hacer esto, necesitamos usar el comando `git checkout`, que se explicará con más detalle más adelante.


Crear branch en GitHub

Para crear una nueva rama en GitHub, debemos hacer clic en el menú desplegable que se encuentra junto al icono . Este menú muestra la rama actual, que generalmente es la rama principal llamada "main" de forma predeterminada. Al hacer clic en el menú, se abrirá un formulario de diálogo donde podemos ingresar el nombre de la nueva rama que queremos crear. También podemos usar este menú para buscar ramas existentes.

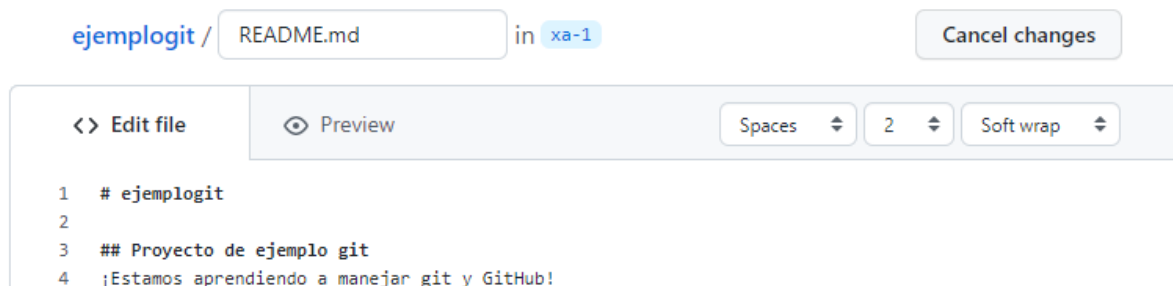


En nuestro caso, el nombre de la nueva rama que queremos crear es "xa-1". Es una buena práctica utilizar un nombre único y descriptivo para identificar la rama, por ejemplo, asociándola a una tarea o funcionalidad específica. Una vez que hemos ingresado el nombre de la nueva rama, hacemos clic en "Create branch". Es importante tener en cuenta que la nueva rama se creará a partir de la rama actual, en este caso "main". Esto significa que la nueva rama "xa-1" será una copia de la rama "main" hasta que se agreguen nuevos cambios a ella.

Hacer cambios desde GitHub

Hacer cambios desde GitHub es muy sencillo. Primero, nos dirigimos a la nueva branch "xa-1" y hacemos clic en el archivo README.md. Luego, hacemos clic en el botón de edición que tiene el ícono del lápiz .

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.



Una vez que hayamos terminado de escribir nuestros cambios, bajamos a la parte inferior de la pantalla. Ahí nos encontraremos con un cuadro de texto donde podemos escribir un mensaje

para nuestro commit. Es importante ser descriptivos en el mensaje y hacer referencia al requerimiento o tarea para proveer información valiosa en el futuro.

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

Finalmente, hacemos clic en el botón “commit changes” para confirmar los cambios realizados en la branch “xa-1”.

Obtener cambios en nuestro local

Para obtener los cambios que se hicieron en el repositorio remoto en nuestro ambiente local, necesitamos dos pasos: primero, obtener los cambios remotos usando el comando "git fetch"; y segundo, cambiar la rama actual de nuestro ambiente local a la nueva rama 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 nos confirmará que hemos cambiado a la nueva rama:

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

Después podemos ejecutar "git status" para revisar el estado de nuestra rama local. Si aparece el mensaje "Your branch is up to date with 'origin/xa-1'", significa que nuestra rama local está sincronizada con la rama remota y no hay cambios pendientes de subir o descargar.

```
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.

```
# ejemplogit

## 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
```

Es importante tener en cuenta que se deben agregar solo los archivos necesarios.

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.

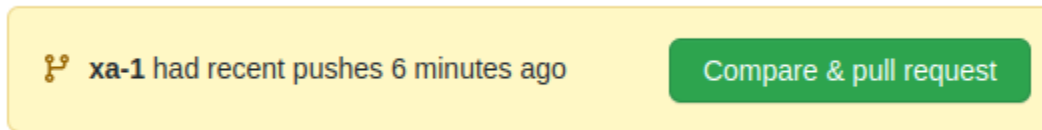


Hacer una Pull Request en GitHub

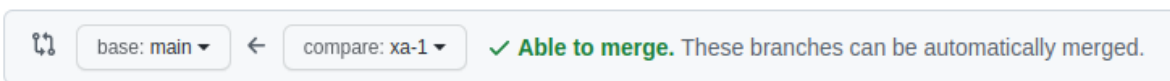
Cuando trabajamos en un proyecto colaborativo, es importante proteger las branches principales para evitar cambios no verificados que puedan generar problemas en el código. Para ello, existen las Pull Requests (PR), que son solicitudes de cambios que enviamos para que otro miembro del equipo revise y apruebe antes de fusionar con la rama principal. En lugar de enviar cambios directamente desde nuestro ambiente local, podemos crear una PR en

GitHub, donde podremos revisar y discutir los cambios realizados antes de integrarlos en la rama principal.

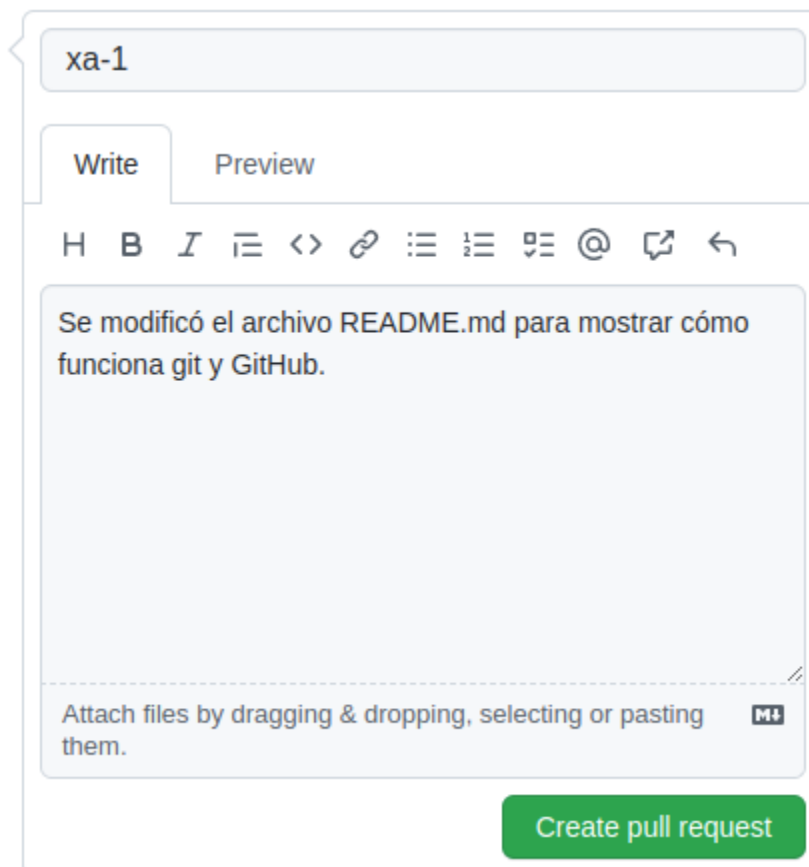
En GitHub aparece un mensaje cuando recientemente una branch recibió modificaciones:



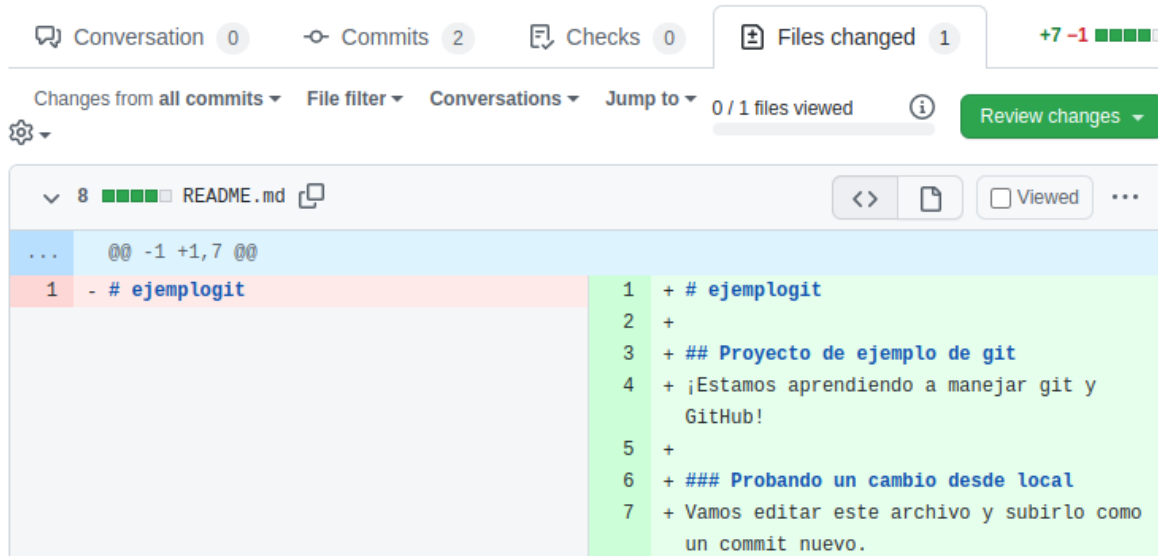
Al hacer clic en “Compare & pull request” seremos redirigidos al formulario para crear una nueva Pull Request. Por defecto, la branch a la que se hace el merge es “main”:



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”

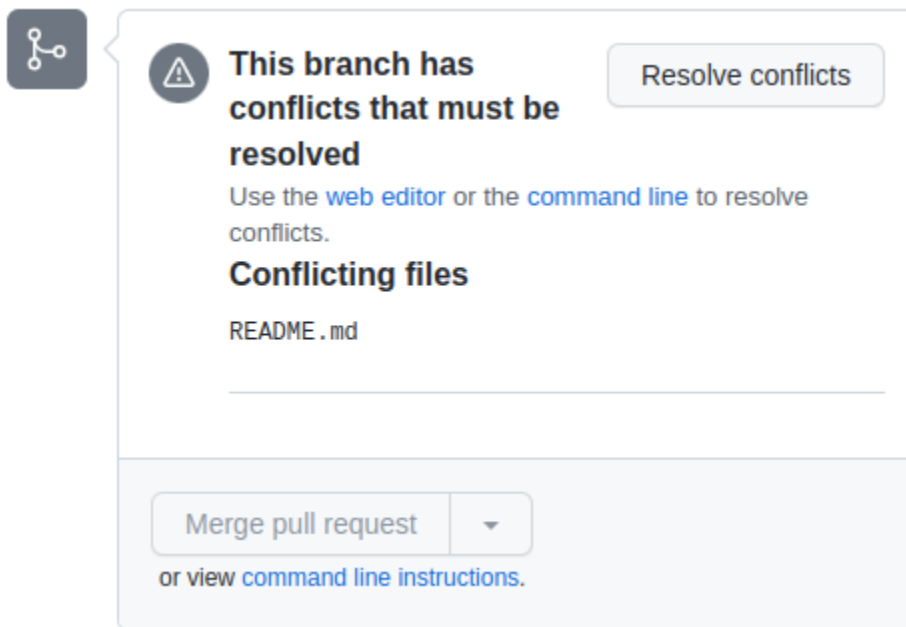
Merge pull request

Resolver conflictos

En nuestro proyecto, 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. Por esta razón, es buena práctica proteger las branches principales para evitar cambios sin verificación por otro compañero.

Si los conflictos son simples, podemos resolverlos directamente en GitHub, pero si son demasiado complejos, será necesario hacerlo en nuestro ambiente local. Para resolver conflictos de código, requerimos mucha atención ya que un desarrollador puede llamar a un método de una forma y otro de otra distinta.

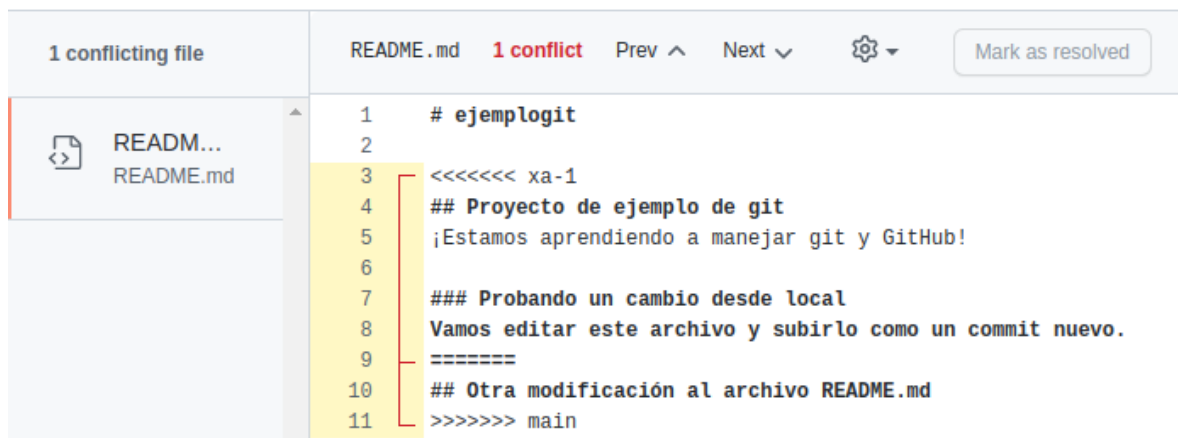
Por ejemplo, si hacemos una Pull Request y un compañero de equipo edita el archivo README.md en la branch principal antes de que se complete la PR, esto generará que en la PR el botón de “Merge pull request” quede desactivado.



En este caso, hacemos clic en el botón “Resolve conflicts” que nos lleva a un editor web donde podemos solucionar los conflictos. Como en este caso es un archivo de texto, es simple darse cuenta de que podemos incluir ambas modificaciones sin romper nada.

xa-1 #1

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



Resolver conflictos requiere mucha atención ya que podría haber diferencias en el código escrito por diferentes desarrolladores, como distintas formas de llamar a un método o utilizar una variable, lo que puede generar conflictos al momento de unir las diferentes ramas del

proyecto

xa-1 #1

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

The screenshot shows a Git conflict resolution window for the file README.md. The interface is divided into two main sections. On the left, there is a sidebar with a file icon and the text 'READM...' and 'README.md'. On the right, the main area displays the content of the file with line numbers 1 through 9. The content is as follows:

```
1 # ejemplodigit
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
```

At the top of the right section, there is a header bar that reads 'README.md 1 conflict'. To the right of this header are navigation buttons: 'Prev ^', 'Next v', a settings gear icon, and a 'Mark as resolved' button.

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.

Ejercicio

1. Hacer un fork del repositorio <https://github.com/SantexGroup/SantexAcademy> en su cuenta personal de GitHub.
2. Clonar el repositorio forkeado en su ambiente local usando SSH:
3. Crea una nueva rama con el nombre "informacion-alumno" usando el comando git branch.
4. Cambia a la rama que acabas de crear usando el comando git checkout.
5. Abre el archivo README.md en tu editor de texto preferido y agrega tu nombre, apellido, nacionalidad y email a la sección de información del proyecto.
6. Agrega los cambios que hiciste al archivo README.md a la zona de preparación usando el comando git add.
7. Crea un nuevo commit con un mensaje descriptivo usando el comando git commit.
8. Sube tus cambios a GitHub usando el comando git push.
9. Crea una Pull Request en GitHub para fusionar tus cambios con la rama principal del repositorio.