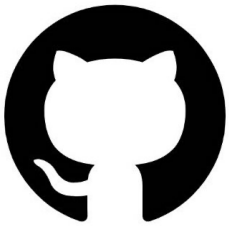
CURSO DE PROGRAMACIÓN FULL STACK

# TUTORIAL: GUÍA DE GIT Y GITHUB CON RAMAS



EGG

## REPASO DE COMANDOS

Veamos un cuadro con los comandos que ya aprendimos

| Comando | Explicación |
|---|---|
| **Git Init** | Para iniciar el trackeo de los archivos de una carpeta. |
| **Git Status** | Nos da toda la información necesaria sobre la rama actual. |
| **Git Add** | Para incluir los cambios del o de los archivos en tu siguiente commit. |
| **Git Commit** | Es como establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario. Es como un punto de guardado en un videojuego. |
| **Git Push** | Enviar archivos incluidos en el commit al repositorio local. |
| **Git Push Origin** | Enviar archivos incluidos en el commit al repositorio remoto. |

### Objetivos de la Guía

En esta guía aprenderemos a:

- Crear una nueva rama
- Clonar una rama
- Moverse entre ramas
- Fusionar ramas
- Borrar una rama
- Utilizar los nuevos comandos - pull request

# RAMAS EN GIT

En la guía anterior vimos que una rama en Git es similar a la rama de un árbol. De manera análoga, una rama de árbol está unida a la parte central del árbol llamada tronco. Si bien las ramas pueden generarse y caerse, el tronco permanece compacto y es la única parte por la cual podemos decir que el árbol está vivo y en pie. De manera similar, una rama en Git es una forma de seguir desarrollando y codificando una nueva función o modificación del software y aún así no afectar la parte principal del proyecto. También vimos que l***a rama principal o predeterminada en Git es la rama maestra o main***.

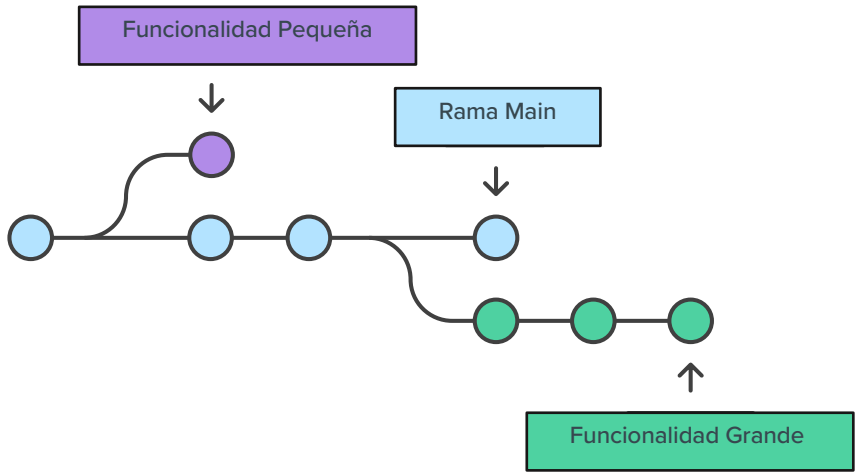
En esta guía vamos a ver como nos pueden ayudar para el trabajo grupal y de que manera podemos crear nuestra propia rama.

## RAMAS PARA EL TRABAJO GRUPAL

Imaginemos que tenemos la siguiente situación, estamos trabajando x personas en un mismo proyecto y cada persona tiene que encargarse de una parte concreta del proyecto, ¿como hacemos para que no estemos todos pisando sobre el mismo proyecto? Acá es donde entran en función las ramas.

Una rama en el trabajo grupal se ve como una **línea independiente de desarrollo**. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación (commit). Puedes concebirlas como una forma de solicitar un nuevo directorio de trabajo o un nuevo entorno de ensayo.

Las ramas son **espacios de desarrollo independientes que emergen de la rama principal** cuando quieres añadir una nueva función o solucionar un error, independientemente de su tamaño, generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que **el código inestable se fusione con el código base principal**, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal.




El diagrama anterior representa un repositorio con dos líneas de desarrollo aisladas, una para una funcionalidad pequeña y otra para una funcionalidad más extensa. Al desarrollarlas en ramas, no solo es posible **trabajar con las dos de forma paralela**, sino que también se evita que el código dudoso se fusione con la rama main.

Explicándolo de otra manera, la rama main va a ser donde este alojado el proyecto final, el proyecto que va a tener todo lo que se trabaje en las demás ramas y las ramas van a ser los lugares en donde cada persona trabajara su parte del proyecto sin afectar al proyecto final que se encuentra en la rama main.

Ahora veremos como se crea una rama propia, como borrarlas y como unir nuestras ramas con la rama principal (rama main)

## GIT BRANCH Y GIT CHECKOUT

Para trabajar con ramas son varios los comandos los que podemos usar, pero dos de los más importantes van a ser **git branch**, que nos va a servir para ver todas las ramas existentes, crear ramas, borrar ramas, renombrar ramas, etc. En cambio, **git checkout** va a ser nuestro aliado para cuando queramos movernos entre ramas.

### VER RAMAS

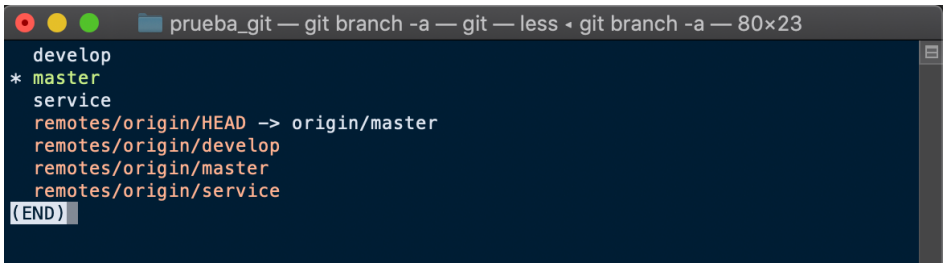
Para ver las ramas en un repositorio Git, ejecuta el comando:

```
git branch
```

Para ver tanto las ramas de seguimiento remoto como las ramas locales, ejecuta el comando:

```
git branch –a
```

Habrá un asterisco (*) junto a la rama en la que te encuentras actualmente.


A veces para salir de esta pantalla deberemos escribir **:wq.**

### MOVERSE ENTRE RAMAS

Para moverse a una rama existente, ejecutamos el comando:

```
git checkout <nombre–de–tu–rama>
```

Generalmente, Git no permitirá moverse a otra rama a menos que el directorio en el que estás trabajando esté limpio, porque podrías perder cualquier cambio en el directorio de trabajo que no esté confirmado.

Podemos obtener una descripción más detallada de las ramas con este otro comando:

```
git show–branch
```

Esto nos muestra todas las ramas del proyecto con sus commits realizados. La salida sería como la de la siguiente imagen.

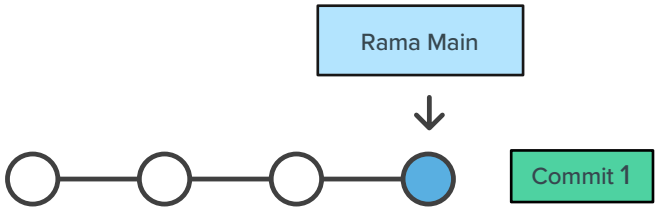

```
[→  Git git:(ramaGit) git show-branch
! [master] Primer commit
 * [ramaGit] Primer commit
--
+* [master] Primer commit
→  Git git:(ramaGit)
```

**Revisemos lo aprendido hasta aquí**

- Ver todas las ramas
- Moverse entre ramas

## CREAR UNA RAMA NUEVA

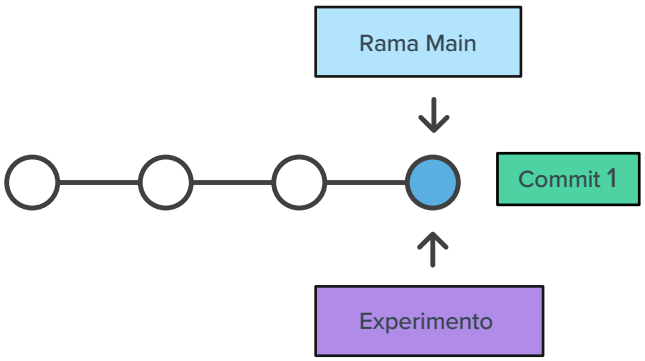
Es importante comprender que las ramas son solo punteros a los commits. Cuando creas una rama, todo lo que Git tiene que hacer es crear un nuevo puntero, no modifica el repositorio de ninguna otra forma. Esto quiere decir que la rama se va a crear con una copia del ultimo commit de la rama main. Si empiezas con un repositorio que tiene este aspecto:




Y, a continuación, creas una rama con el siguiente comando:

```
git branch experimento
```

Nos quedará una rama con el mismo commit y parado en el espacio de tiempo que el ultimo cambio que llegó a la rama main:




Ten en cuenta que este comando **solo crea la nueva rama** tendrás que usar git checkout para moverte a ella y a continuación, utilizar los comandos estándar git add y git commit. Pero, **hay un atajo para crear y moverte** a la nueva rama al mismo tiempo. Puedes pasar la **opción b** (para rama) con git checkout. Los siguientes comandos hacen lo mismo:

```
# Método de 2 pasos

git branch <nombre-de-tu-rama>

git checkout <nombre-de-tu-rama>

# Atajo

git checkout -b <nombre-de-tu-rama>
```

Si nos fijamos no solo creamos una nueva rama local, sino que ahora nos paramos en la nueva rama que creamos.

¿NECESITAS UN **EJEMPLO**?

```
→  Git git:(master) git checkout -b ramaGit
Switched to a new branch 'ramaGit'
→  Git git:(ramaGit)
```

Si corriéramos un git show-branch podríamos ver la rama nueva ya tiene el primer commit que realizamos en la rama main porque como explicamos más arriba, estamos clonando la rama main.

¿NECESITAS UN **EJEMPLO**?

```
→  Git git:(ramaGit) git show-branch
! [master] Primer commit
 * [ramaGit] Primer commit
--
+* [master] Primer commit
→  Git git:(ramaGit)
```

**Revisemos lo aprendido hasta aquí**

- Crear una rama

## CAMBIOS EN LA RAMA

De momento en nuestro ejemplo las dos ramas tenían exactamente el mismo contenido, pero ahora podríamos empezar a hacer cambios en el proyecto ramaGit y sus correspondientes commit y entonces los archivos tendrán códigos diferentes, de modo que puedas ver que al pasar de una rama a otra hay cambios en los archivos.

Al igual que explicamos antes cada vez que quieras subir un cambio a tu branch sitúate en ella y ejecuta los comandos:

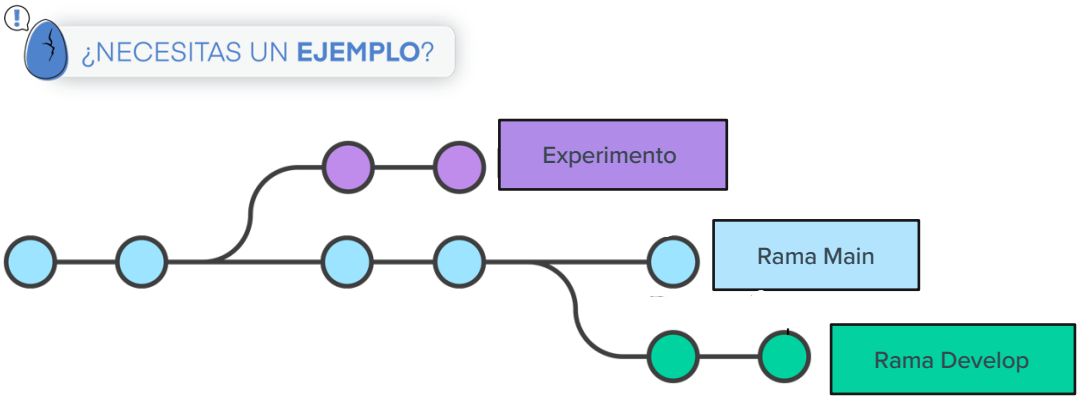
```
git add nombre_del_archivo .
git commit –m “Mensaje de los cambios”
```

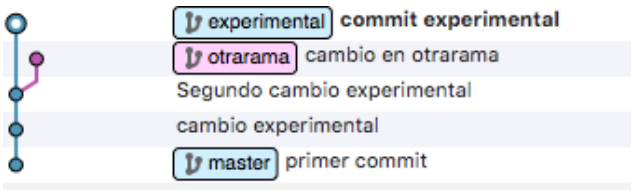
No vamos a hacer un push todavía porque eso lo vamos a explicar más adelante, ya que eso hará que nuestra rama aparezca en el repositorio remoto.

Habiendo hecho un commit en nuestra nueva rama, observarás que al hacer el show-branches te mostrará nuevos datos:

```
[→  Git git:(ramaGit) git show-branch
! [master] Primer commit
 * [ramaGit] Segundo commit
--
 * [ramaGit] Segundo commit
+* [master] Primer commit
```

Como podras ver, el proyecto puede tener varios estados en un momento dado y tú podrás moverte de uno a otro con total libertad y sin tener que cambiar de carpeta ni nada parecido.






**Revisemos lo aprendido hasta aquí**

- Enviar cambios a la rama

## SUBIR UNA RAMA AL REPOSITORIO REMOTO

A pesar de que cuando crear una rama y las puedas ver en tu repositorio local con git branch, las ramas no se publicarán en GitHub. Para que esto ocurra tienes que realizar específicamente la acción de subir una rama determinada.

Para subir una rama en remoto la haces mediante el comando push, indicando el nombre de la rama que deseas subir. Por ejemplo, de esta manera:

```
git push origin <nombre-de-tu-rama>
```

Así estamos haciendo un push, empujando hacia origin (que es el nombre que se suele dar al repositorio remoto).

Si **no quieres poner siempre origin** y el nombre de tu rama en tus push, tienes que sumarle al push anterior, **-u** antes de la palabra origin. Esto hará que puedas poner git push solamente y vaya siempre a esa rama.

Es **importante asegurarse que lo hagamos en una rama nuestra y no en main**, ya que podríamos mandar cambios a la rama main pensando que iban a la nuestra.

Esto sería asi:

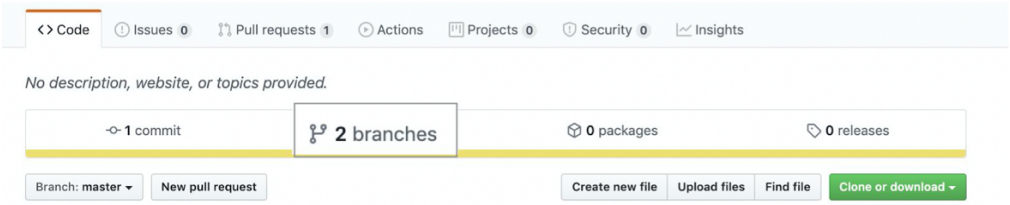
```
git push origin -u <nombre-de-tu-rama>
```

Una vez esto hecho esto podríamos pararnos en nuestra rama y simplemente escribir:

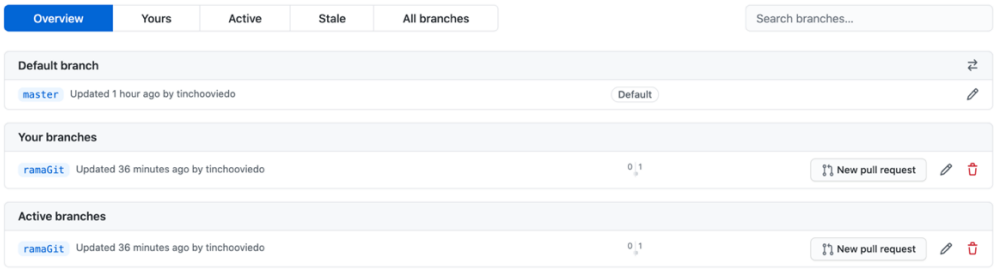
```
git push
```

## ¿Y COMO SE VEN LAS RAMAS EN GITHUB?

Una vez que hagamos el push para ver las ramas, primero iremos a nuestro repositorio y despues a branches:




Y ahí podremos ver las dos ramas



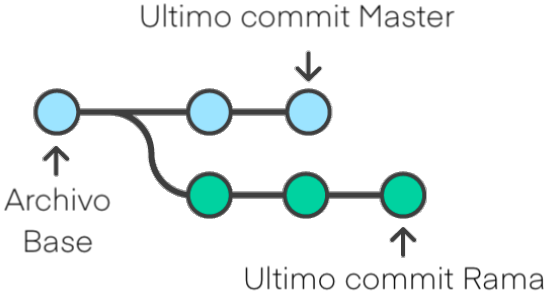

Podes subir tanto commits como creas convenientes a tu rama antes de fusionar tus cambios con la rama main, siempre es **mejor pequeños y frecuentes cambios que pocos y grandes.**

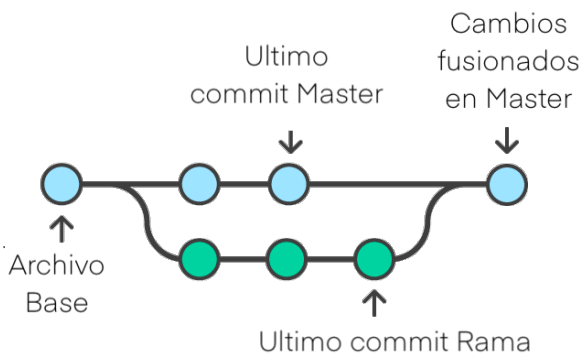
**Revisemos lo aprendido hasta aquí**

- Subir una rama al repositorio remoto

## FUSIONAR RAMAS

A medida que crees ramas y cambies el estado de las carpetas o archivos de tu proyecto empezará a divergir de una rama a otra. Llegará el momento en el que te interese fusionar ramas para poder incorporar el trabajo realizado a la rama main.




El proceso de fusionado se conoce como **merge** y puede llegar a ser muy simple o más complejo si se encuentran cambios que Git no pueda procesar de manera automática. Git para procesar los *merge* usa un antecesor común y comprueba los cambios que se han introducido al proyecto desde entonces, combinando el código de ambas ramas.

Para hacer un merge nos situamos en una rama, en este caso la "main", y decimos con qué otra rama se debe fusionar el código.

El siguiente comando, lanzado desde la rama "main", permite fusionarla con la rama "ramaGit".

```
git merge <nombre-de-tu-rama>
```

Un *merge* necesita un mensaje, igual que ocurre con los *commit*, por lo que al realizar ese comando se abrirá "Vim" (o cualquier otro editor de consola que tengas configurado) para que introduzcas los comentarios que juzgues oportuno. Salir de Vim lo consigues pulsando la tecla ESC y luego escribiendo **:q** y pulsando ENTER para aceptar ese comando. Esta operativa de indicar el mensaje se puede resumir con el comando:

```
git merge ramagit -m "Esto es un merge con mensaje"
```

Luego podremos comprobar que nuestra rama main tiene todo el código nuevo de la ramaGit y podremos hacer nuevos commits en main para seguir el desarrollo de nuestro proyecto ya con la rama principal, si es nuestro deseo.

**Revisemos lo aprendido hasta aquí**

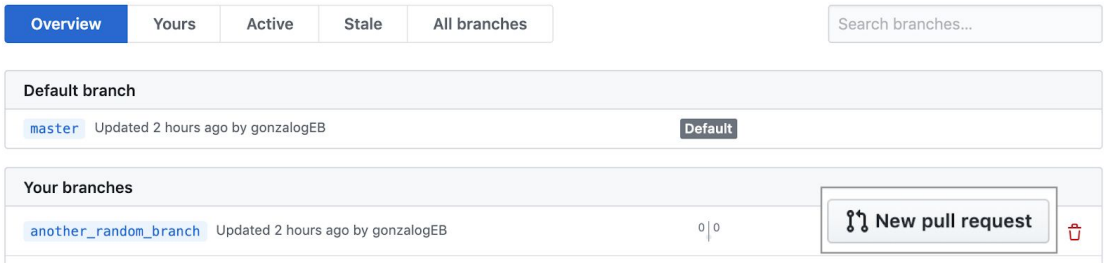
- Fusionar una rama

# PULL REQUEST

Previamente habíamos fusionado nuestras ramas a través del comando **merge**, pero GitHub nos permite fusionar nuestras ramas y además ver los cambios que hay entre una rama y otra, gracias al **Pull Request.**

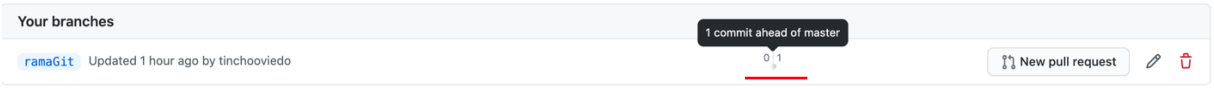
Vamos a pararnos de nuevo en una etapa en donde no hemos mergeado las ramas. Hasta ese punto habíamos logrado independizar nuestros cambios de los del resto del equipo, pero se acercaba la hora de publicar nuestros cambios y surge la necesidad de conocer y/o validar cuan diferente es nuestra versión y de ver que todo está bien fusionar esos cambios. Aquí es donde la herramienta de pull request viene al rescate.

Para crear un pull request debemos ir a la sección de branches, buscar nuestro branch y clickear en el botón **New pull request**.

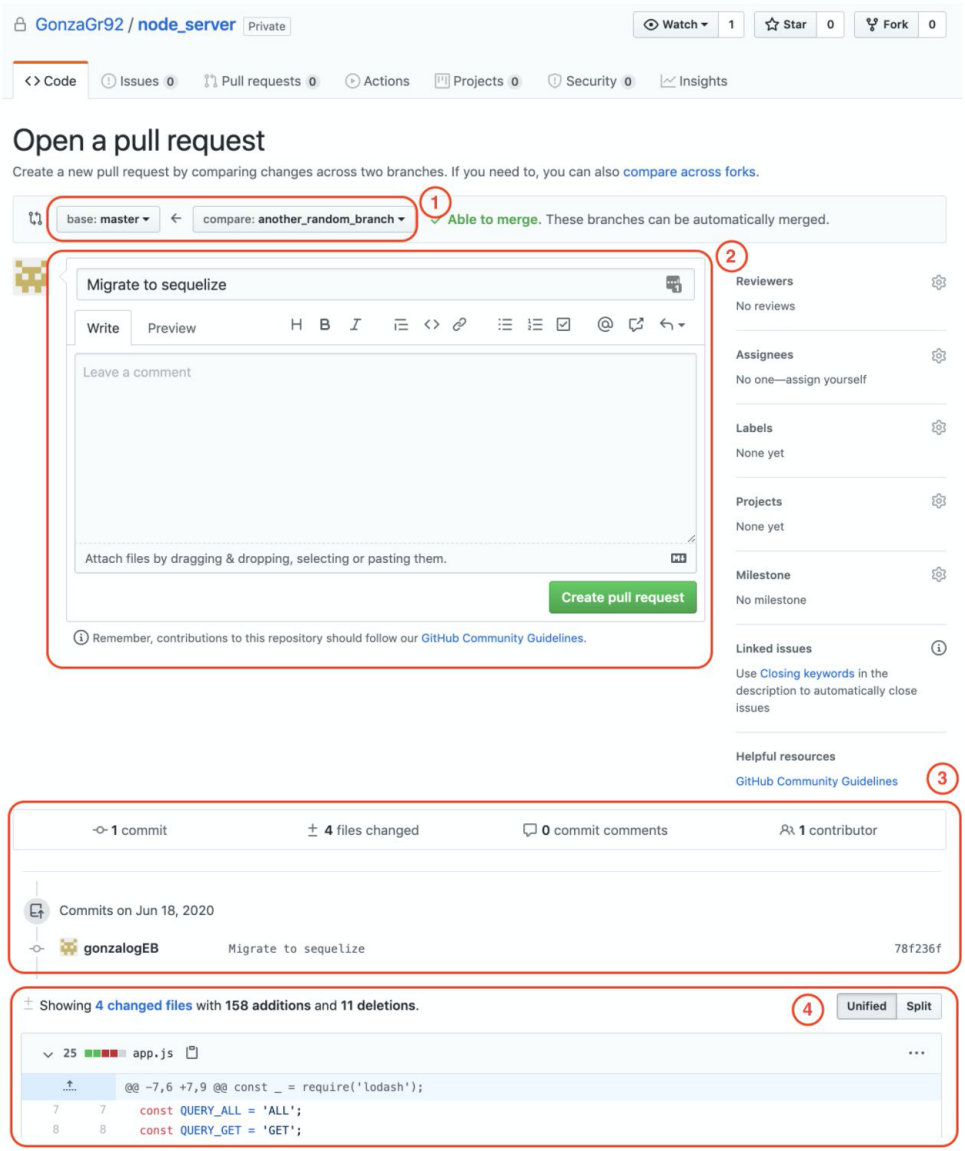



Antes de hacer nuestro pull request, podemos ver, cuantos commits (cambios) son los que separan a otra rama de main. Si nos fijamos nos salen dos ceros, pero si main estuviera un commit por delante de alguna rama saldrían un uno en el cero de la izquierda y asi se incrementa el numero según la cantidad de commits que este por delante main. Ahora, si el numero estuviera a la derecha, sería que la otra rama está x commits por delante main.

Aca podemos ver un ejemplo con ramaGit:




Como podemos ver ramaGit está un commit por delante de main, por lo que si mergeamos las ramas, sería solo un commit el que se le aplicaría a main.




Nos mostrará algo similar a lo siguiente.

Referencias:

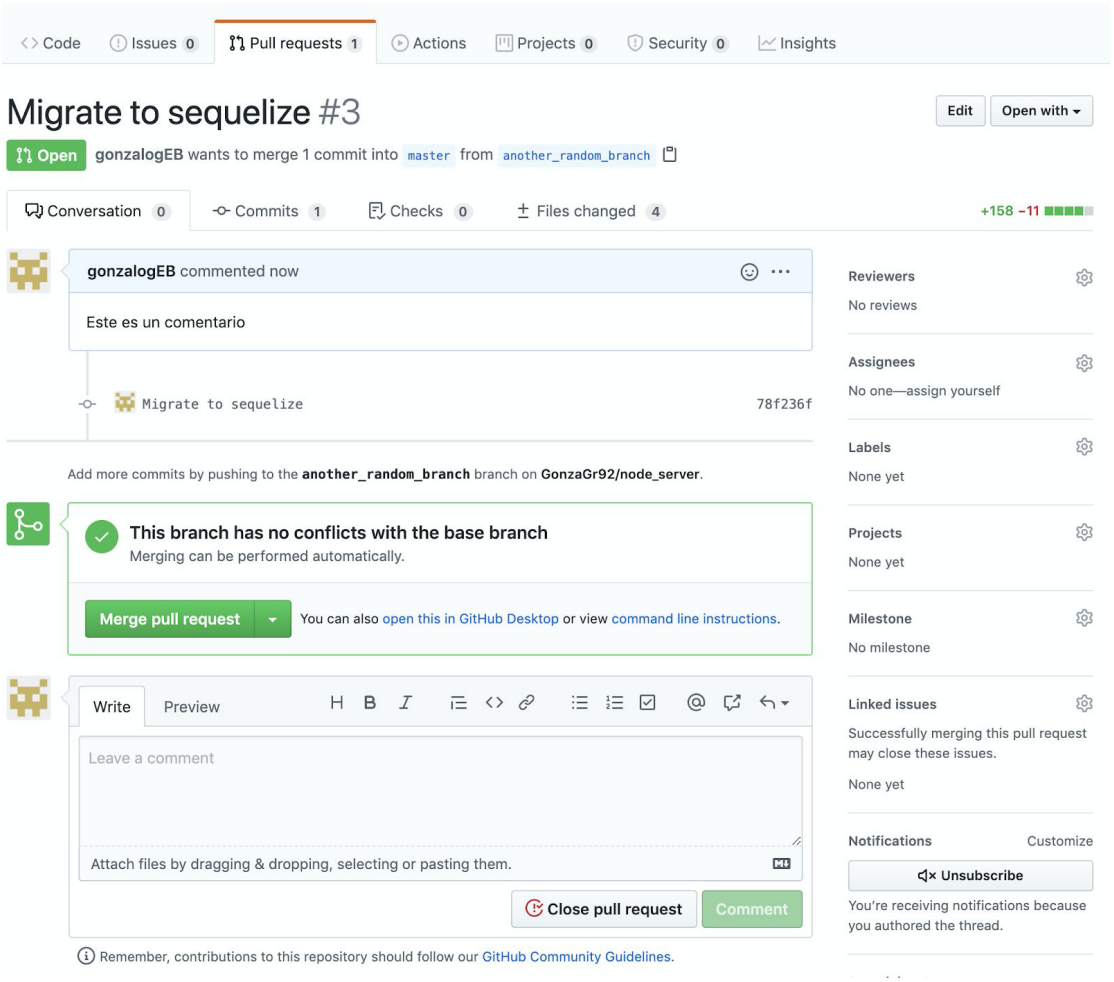
1. A la izquierda se selecciona la rama de destino a la cual vamos a querer mergear los cambios, a la derecha nuestra rama actual. Siempre podemos crear un pull request y cambiar las ramas que queremos mergear.

2. En esta sección podremos poner un titulo informativo de que se tratan nuestros cambios y una descripción como documentación adicional. Detalle de los cambios propuestos, test plan, etc.

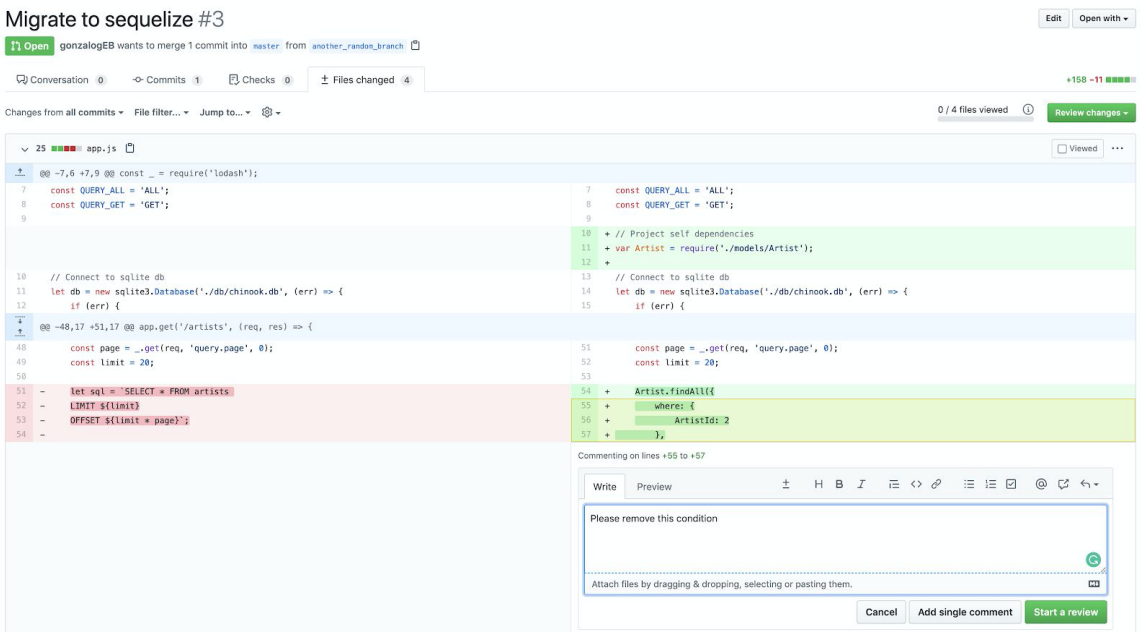
3. En esta sección nos muestra un resumen de los commits y archivos modificados.

4. En la ultima sección nos muestra el detalle de los archivos modificados. Para visualizar los cambios podemos alternar entre los modos de vista “Unified” y “Split”

Hasta este momento aún no hemos creado nada, solo estamos viendo un resumen previo, para continuar clickeamos en el botón “Create pull request”. A continuación, veremos el pull request creado de la siguiente manera.

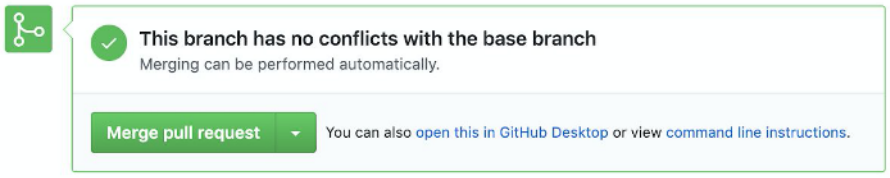



Por otro lado, podemos usar la pestaña de “Files changed” para hacer code review.

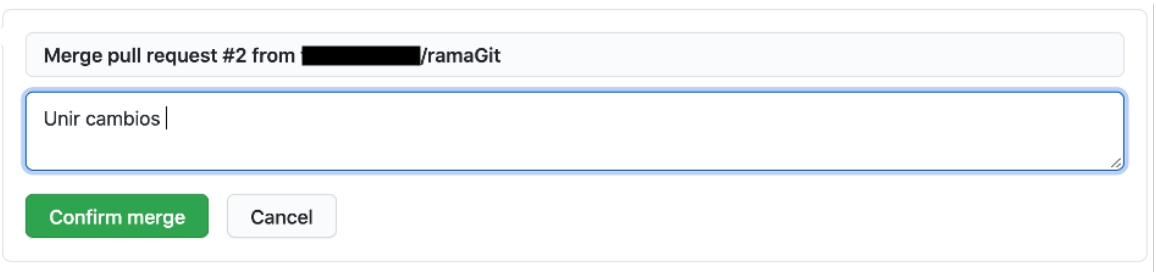



Si detectamos alguna línea de código que requiera cambios puedes clickear sobre ella y agregar un comentario para que el autor del pull request lo modifique.  No es necesario volver a crear un nuevo pull request para actualizar los cambios, simplemente haciendo un commit sobre el branch es suficiente, GitHub toma los cambios y actualiza el pull request automáticamente.

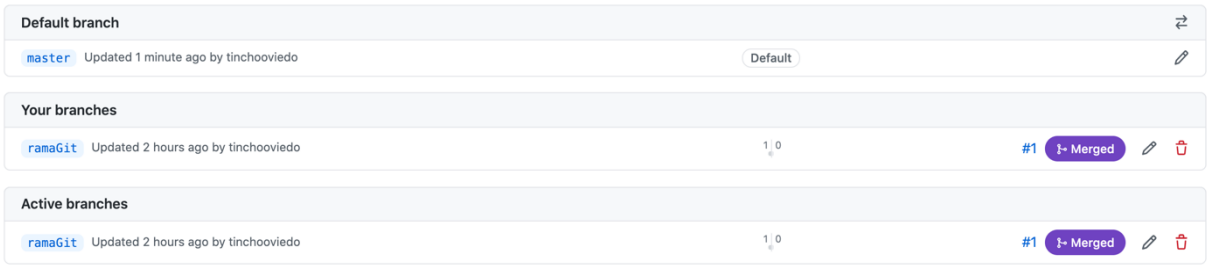
Si esta todo bien y no hay conflictos podemos mergear nuestro branch a main clickeando en el botón “Merge pull request” y de eso modo finaliza el ciclo del branch.




Finalmente tenemos la opción de aprobar los cambios para que el autor del pull request mergee su cambio.




Una vez que unamos los cambios en nuestra ramas nos va a salir que la rama ha sido mergeada.




## Revisemos lo aprendido hasta aquí

- Utilizar el pull request para fusionar ramas

# DESCARGAR CONTENIDO DESDE UN REPOSITORIO REMOTO

En la guía anterior vimos como clonar un repositorio, pero imaginemos que ya estamos trabajando en dicho repositorio con un grupo de personas. A veces ocurre que las personas del equipo generan sus propias ramas en remoto, donde trabajarán su parte del proyecto por ejemplo cuando han sido creadas por otros usuarios y subidas al hosting de repositorios.

Ahora supongamos necesitamos acceder a las ramas y su contenido en local **para verificar los cambios o continuar el trabajo**. En principio esas ramas en remoto creadas por otros usuarios no están disponibles para nosotros en local, pero las podemos descargar.

Para esto usaremos el comando **git pull**

## GIT PULL

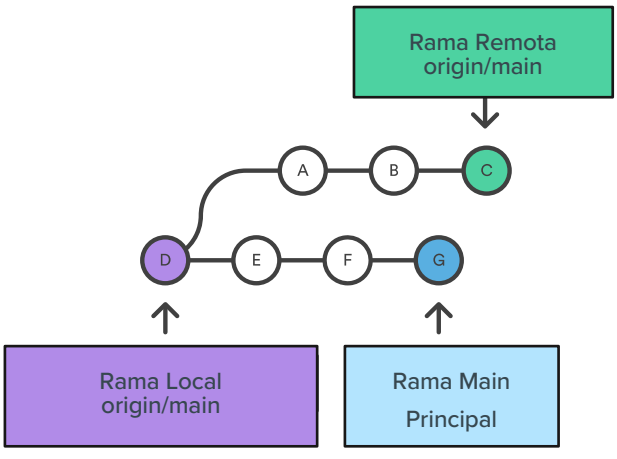
Git pull se utiliza para mantener el repositorio actualizado con los cambios que haya habido. No siempre, pero en muchos casos hay más de una persona trabajando en un repositorio, por lo que cuando una persona hace cambios en un repositorio remoto con un git push, nuestro repositorio local estará desactualizado y tendremos que traernos los cambios que se hayan hecho. Supongamos que un compañero nuestro hace un cambio y nosotros debemos seguir trabajando sobre ese cambio, para esto nos sirve usar el git pull

Git pull comprueba si hay cambios en el repositorio remoto y, en caso de que los haya, se trae esos archivos a tu repositorio local y actualiza tu espacio de trabajo (tu IDE, tus archivos).

Es uno de los cuatro comandos que solicita interacción de red por Git. Por default, git pull hace dos cosas.

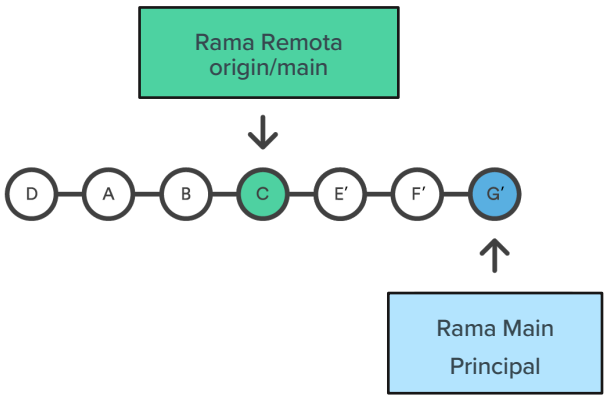
1. Actualiza la rama de trabajo actual (la rama a la que se ha cambiado actualmente)
2. Actualiza las referencias de rama remota para todas las demás ramas.

git pull recupera (git fetch) los nuevos commits y las fusiona (git merge) en tu rama local.




Nuestra rama está en el commit inicial que es commit D, la rama de otra persona está en el commit C y el Main está en el commit G.

En este caso, git pull descargará todos los cambios desde el punto de separación de la rama local y la rama principal. En el ejemplo de arriba, ese punto es E. El comando git pull recuperará los commits remotos divergentes, que son A, B y C. A continuación, el proceso de incorporación de cambios creará otro de fusión local que incluya el contenido de las nuevas confirmaciones remotas divergentes.




Ahora podemos ver que la incorporación mediante cambio de base no ha creado la confirmación H, sino que el cambio de base ha copiado los commits remotos A, B y C y ha reescrito los commits locales E, F y G para que aparezcan después de ellas en el historial de commits principales o de origen locales.

## ¿QUÉ ES UNA RAMA REMOTA?

Las ramas remotas son referencias al estado de las ramas en tus repositorios remotos. Son ramas locales que no puedes mover; se mueven automáticamente cuando estableces comunicaciones en la red. Las ramas remotas funcionan como marcadores, para recordarte en qué estado se encontraban tus repositorios remotos la última vez que conectaste con ellos.

Suelen referenciarse como (remoto)/(rama). Por ejemplo, si quieres saber cómo estaba la rama main en el remoto origin, puedes revisar la rama **origin/main**. O si estás trabajando en un problema con un compañero y este envía (push) una rama **develop**, tú tendrás tu propia rama de trabajo local **develop**; pero la rama en el servidor apuntará a la última confirmación (commit) en la rama **origin/develop.**

Esto puede ser un tanto confuso, pero intentemos aclararlo con un ejemplo. Supongamos que tienes un servidor Git en tu red, en **git.ourcompany.com**. Si haces un clon desde ahí, Git automáticamente lo denominará origin, traerá (pull) sus datos, creará un apuntador hacia donde esté en ese momento su rama main y denominará la copia local origin/main. Git te proporcionará también tu propia rama main, apuntando al mismo lugar que la rama main de origin; de manera que tengas donde trabajar.

## SINTAXIS GIT PULL

La síntaxis de este comando es el siguiente:

```
git pull <nombre-remoto> <nombre-de-tu-rama>
```

**Nombre-remoto:** es el nombre de tu repositorio remoto. Por ejemplo: origin.

**Nombre-de-tu-rama:** es el nombre de tu rama. Por ejemplo: develop.

Por ejemplo:

```
git pull origin ramaGit
```

**Nota:**

Si tienes cambios no confirmados, la parte de fusión del comando git pull fallará y tu rama local quedará intacta.

Por lo tanto, *siempre deberías confirmar tus cambios en una rama antes de actualizar* nuevas confirmaciones de un repositorio remoto.

**Revisemos lo aprendido hasta aquí**

- Utilizar el comando Git Pull
- Descargar ramas del repositorio remoto

## BORRAR UNA RAMA

En ocasiones puede ser necesario eliminar una rama del repositorio, por ejemplo, porque nos hayamos equivocado en el nombre al crearla. Aquí la operativa puede ser diferente, dependiendo de si hemos subido ya esa rama a remoto o si todavía solamente está en local.

### BORRADO DE LA RAMA EN LOCAL

Esto lo conseguimos con el comando git branch, solamente que ahora usamos la opción **-d** para indicar que esa rama queremos borrarla.

```
git branch -d <rama-a-borrar>
```

Sin embargo, puede que esta acción no nos funcione porque hayamos hecho cambios que no se hayan salvado en el repositorio remoto, o no se hayan fusionado con otras ramas. En el caso que queramos forzar el borrado de la rama, para eliminarla independientemente de si se ha hecho el push o el merge, tendrás que usar la opción **-D**.

```
git branch -D <rama-a-borrar>
```

Debes prestar especial atención a esta opción "-D", ya que al eliminar de este modo puede haber cambios que ya no se puedan recuperar. Como puedes apreciar, es bastante fácil de confundir con "-d", opción más segura, ya que no permite borrado de ramas en situaciones donde se pueda perder código.

### ELIMINAR UN BRANCH EN REMOTO

Si la rama que queremos eliminar está en el repositorio remoto, la operativa es un poco diferente. Tenemos que hacer un push, indicando la opción --delete, seguida de la rama que se desea borrar.

```
git push origin --delete <rama-a-borrar>
```

**Revisemos lo aprendido hasta aquí**

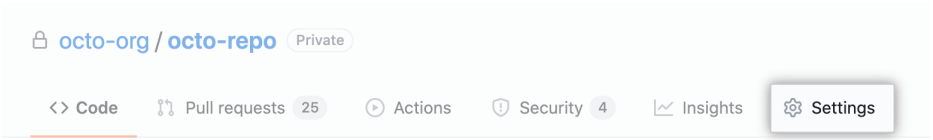
- Eliminar ramas del repositorio local y remoto

# INVITAR COLABORADORES A UN REPOSITORIO PERSONAL

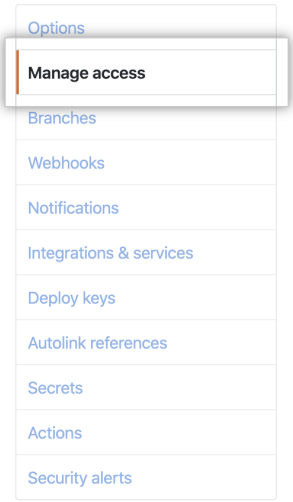
Nosotros vimos como clonar un repositorio para poder usar su código y si quisiéramos hacer cambios y subir esos cambios. Todos los usuarios de GitHub pueden ver y clonar tu repositorio, siempre y cuando sea un repositorio público. Pero, no todas las personas que clonan tu repositorio pueden subir sus cambios, ya que GitHub entiende que los repositorios son de nuestra propiedad y somos los únicos que podemos modificarlo.

Ahora, como hacemos cuando queremos que varias personas trabajen en un mismo repositorio y queremos que GitHub les deje subir esos cambios. Para ese dilema GitHub nos deja invitar colaboradores a nuestro proyecto. Esto se hará de la siguiente manera:

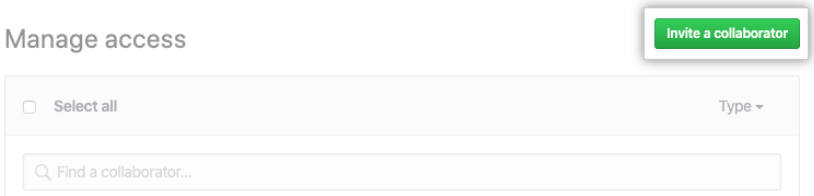
1. Solicita el nombre de usuario de la persona que estás invitando como colaboradora.
2. En GitHub, visita la página principal del repositorio.
3. Debajo de tu nombre de repositorio, da clic en **Configuración.**




4. En la barra lateral izquierda, da clic en Administrar acceso.

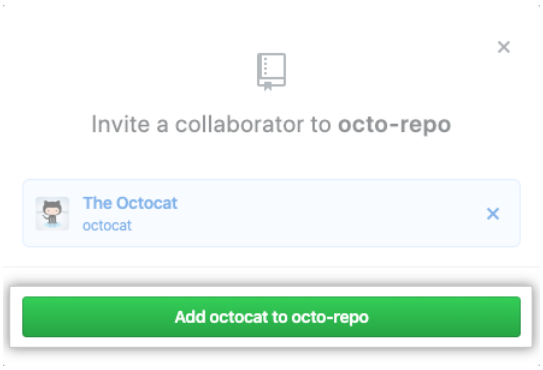



5. Da clic en Invitar un colaborador.




6. En el campo de búsqueda, comienza a teclear el nombre de la persona que quieres invitar, luego da clic en un nombre de la lista de resultados.

7. Da clic en **Añadir a nombreRepositorio**.



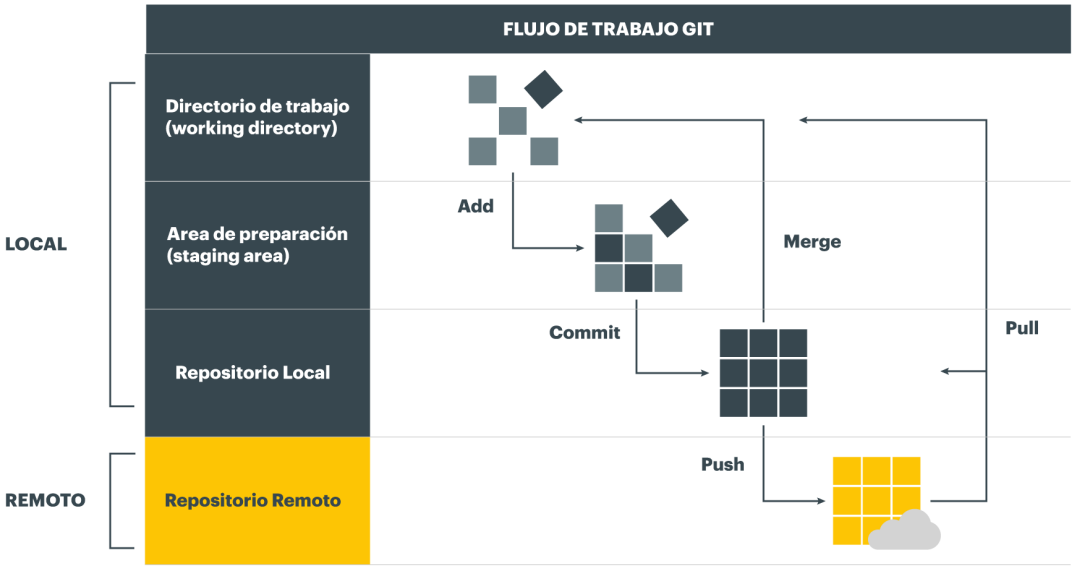

8. El usuario recibirá un correo electrónico invitándolo al repositorio. Una vez que acepte la invitación, tendrá acceso de colaborador a tu repositorio. Las invitaciones pendientes caducarán después de 7 días. Esto restablecerá cualquier licencia sin reclamar.

**Revisemos lo aprendido hasta aquí**

- Invitar colaboradores a tu repositorio

## FLUJO DE TRABAJO GIT

En la guía anterior vimos un flujo de trabajo de Git con GitHub, pero en esta guía aprendimos nuevos comandos que se sumarán a este flujo de trabajo. Así que veremos el mismo diagrama con estos nuevos comandos.



## EJERCICIOS DE APRENDIZAJE

Ahora es momento de poner en práctica todo lo visto en la guía.

**VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.**

**1.** Para el siguiente ejercicio, van a tener que trabajar en equipo, con sus compañeros de mesa.

a) El facilitador de cada equipo debe crear un repositorio público con el nombre practica_github seleccionando la opción Initialize this repository with a README.

b) Una vez creado el repositorio el facilitador debe invitar a los integrantes de su mesa al mismo. Clickear en el botón Invite a collaborator y buscar a los miembros de su mesa por username o email.

c) Cada miembro debe aceptar la invitación al repositorio. Checkear la invitación el email y clickear en View Invitation.

d) Clonar el repositorio. Cada miembro del equipo debe clonar el repositorio con el archivo ReadMe. Luego de aceptar la invitación github te redirige al repositorio.

e) Cada miembro de la mesa, incluido el facilitador, debe crear su propia rama para trabajar sobre el archivo ReadMe

f) Ahora cada miembro de la mesa debe incluir su nombre en el archivo ReadMe de manera local y subirlo a su rama.

g) Cuando todos los miembros de la mesa han agregado su nombre al archivo ReadMe, de uno en uno, ir uniendo en la rama main todos vuestros cambios. Al final les debería quedar un archivo ReadMe con todos sus nombres.

**2.** Ahora van a continuar trabajando como mesa. Vuestra tarea ahora es que cada miembro de la mesa, incluido el facilitador, debe crear su branch y crear una de las siguientes clases: Gato, Perro, Caballo, Conejo, Pájaro y Pato. Cada uno le va a poner los atributos que desee.

El facilitador va a tener que crear el repositorio y subir un proyecto de Java vacio para que los miembros de la mesa puedan clonar y crear su clase. Una vez que cada miembro haya creado su clase en su respectiva rama, deberán unir todas las clases en la rama main, para que quede el proyecto final.

## GLOSARIO

1. Puntero: **Un puntero no es más que una variable, en la cual se almacena una dirección de memoria**. Al ser una dirección de memoria, le podemos decir a un puntero que en ese lugar donde apunta queremos almacenar un valor, por ejemplo, un número. En el caso de git la rama va a almacenar la dirección de memoria de un commit de la rama main.