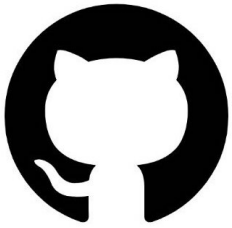
CURSO DE PROGRAMACIÓN FULL STACK

TUTORIAL: GUÍA DE GIT Y GITHUB CON RAMAS



EGG

GUÍA DE GIT Y GITHUB CON RAMAS

Un repaso de los comandos que ya aprendimos

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.

RAMAS EN GIT

En la guía anterior vimos que las ramas nos ayudan a trabajar en conjunto y evitar problemas y colisionar código permanentemente. Nosotros podemos crear nuestra propia rama del proyecto y hacer todos los cambios que necesites, y al final del proceso crear un pull request para mergear (juntar tus cambios) con la rama principal, main o master.

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

GIT CHECKOUT Y GIT BRANCH

En la guía anterior vimos el comando **git branch,** este nos servía para ver en que rama nos encontramos parados antes de un commit o después de un commit, pero este también es el que usaremos principalmente para trabajar con la creación de ramas, borrado de ramas y demás. Sin embargo, no es el único comando para la operativa que veremos en este artículo, ya que existen otros subcomandos de Git útiles y necesarios para trabajar con ramas, como checkout para crear y moverse entre ramas

Volvamos un segundo al comando git branch, esto nos dará el listado de ramas que tengamos en un proyecto. Pero hay que advertir que las ramas de un repositorio local pueden ser distintas de las ramas de un repositorio remoto. Por ejemplo, cuando clonas un repositorio de GitHub generalmente estás clonando únicamente la rama master y no todas las ramas que se hayan creado a lo largo del tiempo. Otro ejemplo es cuando creas una rama en tu repositorio local. En este caso la rama la tendrás simplemente en tu proyecto local y no se subirá al repositorio remoto hasta que no lo especifiques.

CREAR UNA RAMA NUEVA

El procedimiento para crear una nueva rama es bien simple. Usando el comando checkout seguido del nombre de la rama que queremos crear. El guion b lo que hace es crear la rama y cambiar a esa rama nueva.

```
git checkout -b nombre_de_tu_branch
```

Si nos fijamos nos solo creamos una nueva rama local, sino que ahora nos paramos en la nueva rama que creamos.

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

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)
```

Como podemos ver la rama nueva ya tiene el primer commit que realizamos en la rama master porque como explicamos más arriba, estamos clonando la rama master.

PASAR DE UNA RAMA A OTRA

Para moverse entre ramas usamos el comando git checkout seguido del nombre de la rama que queremos que sea la activa.

```
git checkout nombre_de_tu_branch
```

Esto se vería así:

```
➜  Git git:(ramaGit) git checkout master
Switched to branch 'master'
➜  Git git:(master) git checkout ramaGit
Switched to branch 'ramaGit'
➜  Git git:(ramaGit)
```

Esta sencilla operación tiene mucha potencia, porque nos cambiará automáticamente todos los archivos de nuestro proyecto, los de todas las carpetas, para que tengan el contenido en el que se encuentren en la correspondiente 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 (punto(.) en lugar del nombre si quieres agregar todos cambiados)
```

```
git commit -m “Mensaje de los cambios”
```

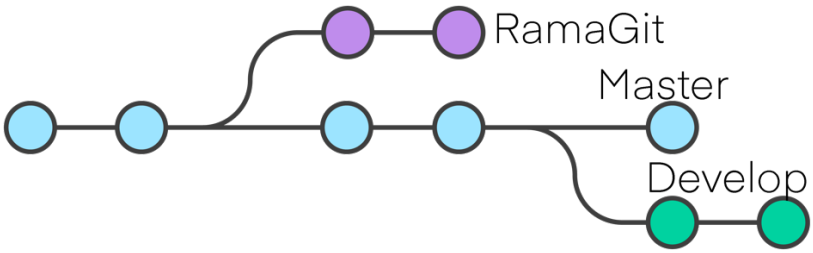
No vamos a hacer un push todavía porque eso lo vamos a explicar más adelante, ya que eso hará que nuestra rama aparezca en el repositorio remoto.

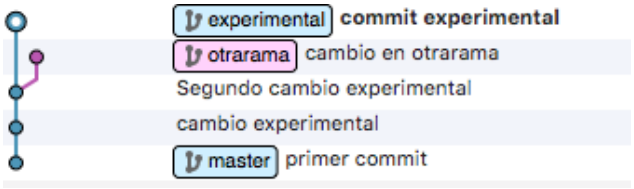
Habiendo hecho un commit en nuestra nueva rama, observarás que al hacer el show-branches te mostrará nuevos datos:

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

Si te dedicas a editar tus ficheros, crear nuevos archivos y demás en las distintas ramas entonces podrás observar que al moverte de una a otra con *checkout* el proyecto cambia automáticamente en tu editor, mostrando el estado actual en cada una de las ramas donde te estás situando. Es algo divertido y, si eres nuevo en Git verás que es una magia que resulta bastante sorprendente.

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.





SUBIR UNA RAMA AL REPOSITORIO REMOTO

Como habíamos dicho anteriormente, por mucho que hagas la operativa descrita para crear ramas en tu ordenador, y las puedas ver en tu repositorio local con git branch, las ramas no se publicarán en Github o cualquier otro hosting de repositorios remoto. Para que esto ocurra tienes que realizar específicamente la acción de subir una rama determinada.

La operativa de publicar 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_branch
```

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 master, ya que podríamos mandar cambios a la rama master pensando que iban a la nuestra.

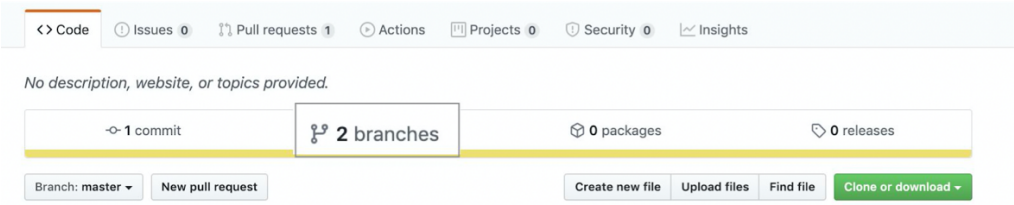
Esto sería asi:

```
git push -u origin nombre_de_tu_branch
```

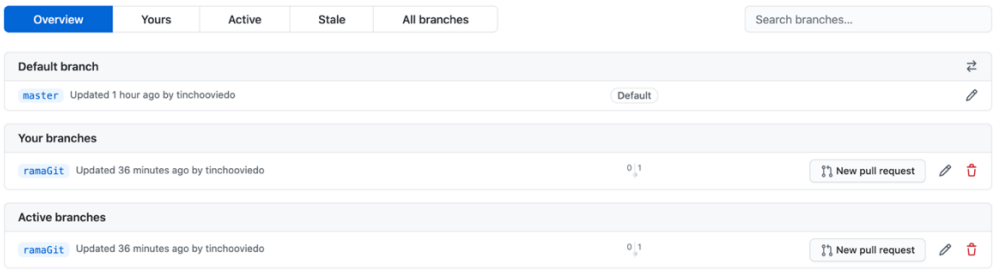
Una vez esto hecho esto podríamos pararnos en nuestra rama y simplemente escribir:

```
git push
```

Una vez que hagamos para ver las ramas, primero iremos a branches:



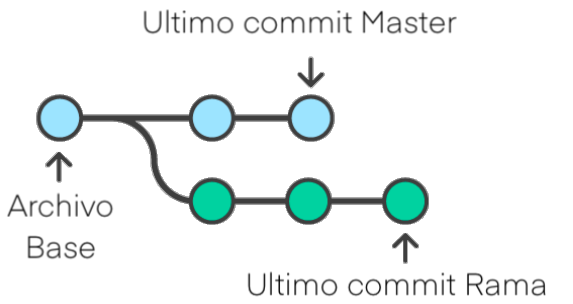
Y ahí podremos ver las dos ramas

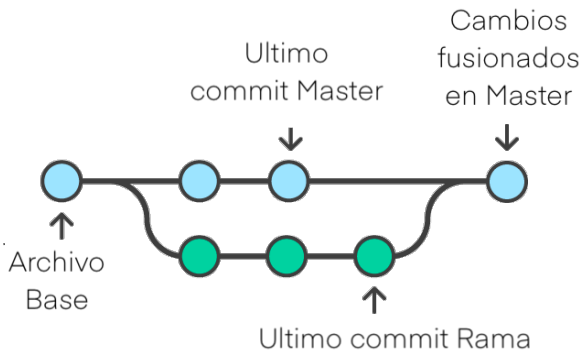


Puedes subir tanto commits creas convenientes a tu branch antes de mergear a master, siempre es mejor pequeños y frecuentes cambios que pocos y grandes.

FUSIONAR RAMAS

A medida que crees ramas y cambies el estado de las carpetas o archivos 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 master.





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 "master", y decimos con qué otra rama se debe fusionar el código.

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

```
git merge ramaGit
```

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”
```

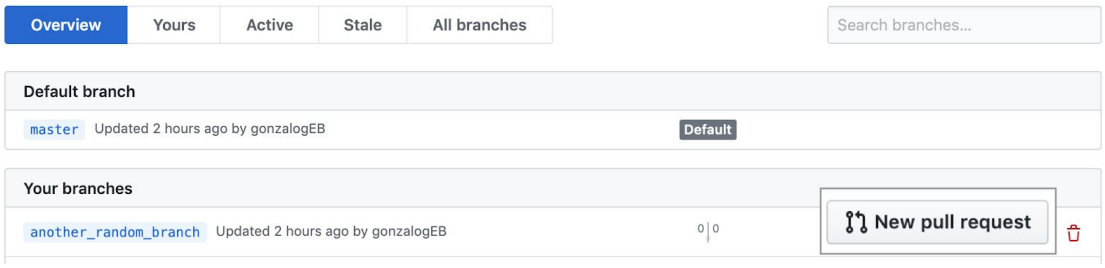
Luego podremos comprobar que nuestra rama master tiene todo el código nuevo de la ramaGit y podremos hacer nuevos commits en master para seguir el desarrollo de nuestro proyecto ya con la rama principal, si es nuestro deseo.

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**

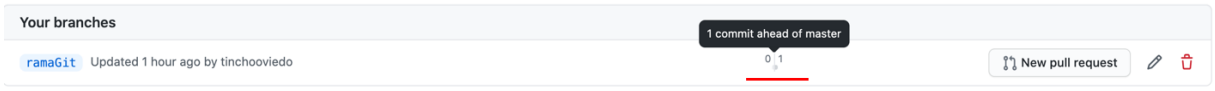
Vamos a pararnos de nuevo en una etapa en donde no hemos mergeado las ramas. Hasta ese punto habíamos logrado independizar nuestros cambios de los del resto del equipo, pero se acercaba la hora de publicar nuestros cambios y surge la necesidad de conocer y/o validar cuan diferente es nuestra versión 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**.

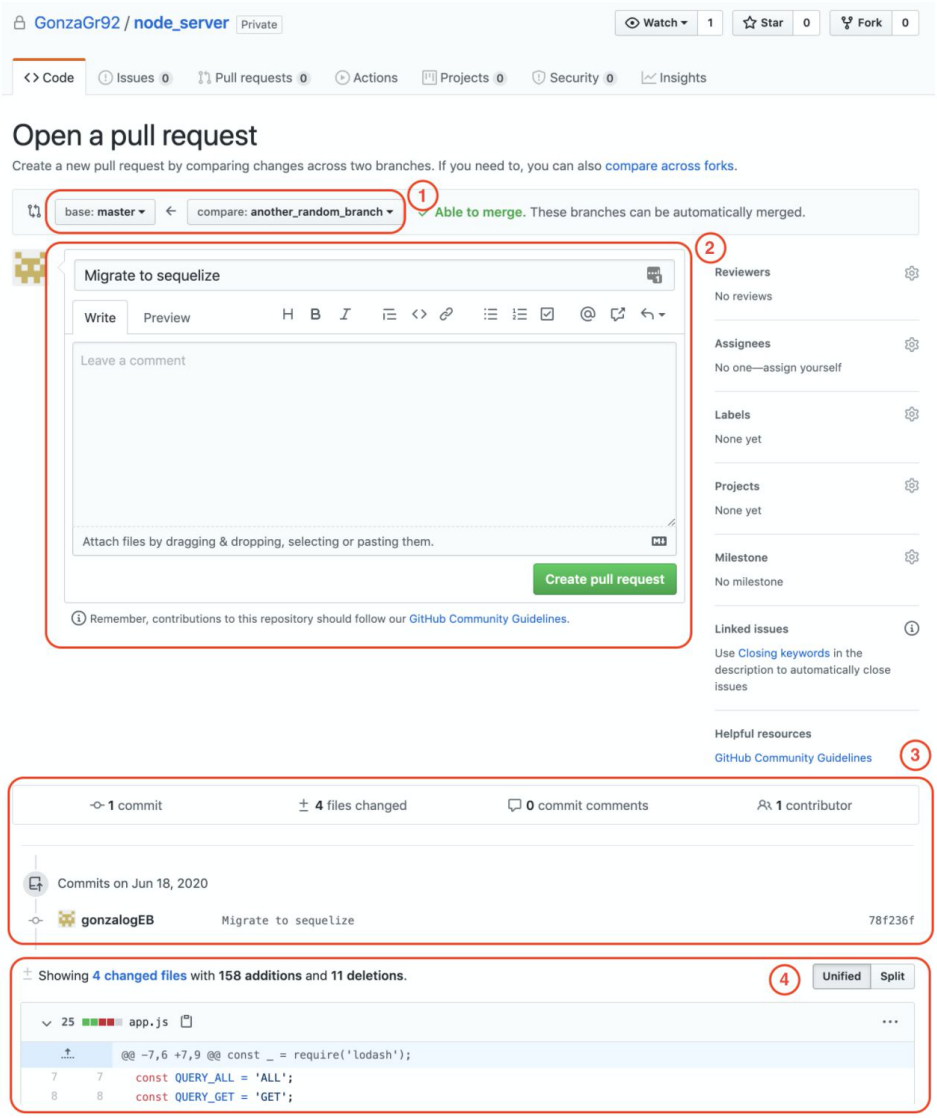


Antes de hacer nuestro pull request, podemos ver, cuantos commits (cambios) son los que separan a otra rama de master. Si nos fijamos nos salen dos ceros, pero si master estuviera un commit por delante de alguna rama saldrían un uno en el cero de la izquierda y asi se incrementa el numero según la cantidad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

Aca podemos ver un ejemplo con ramaGit:



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

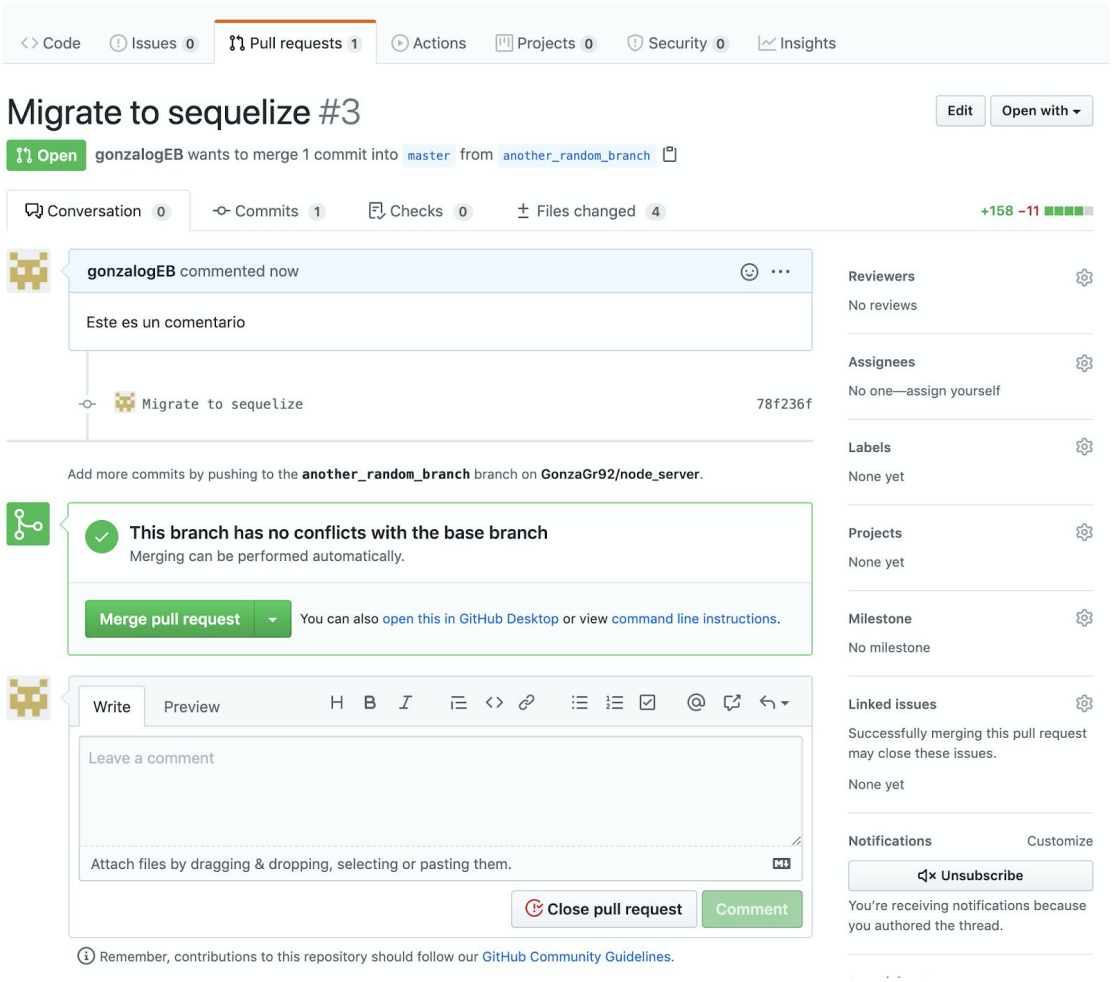


Nos mostrará algo similar a lo siguiente.

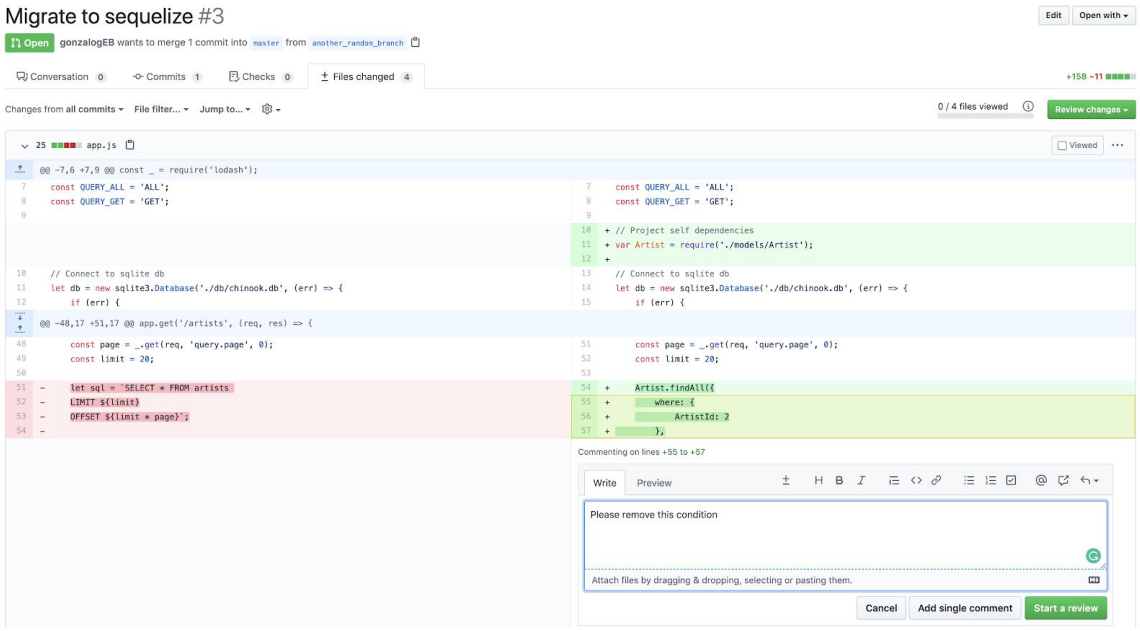
Referencias:

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.

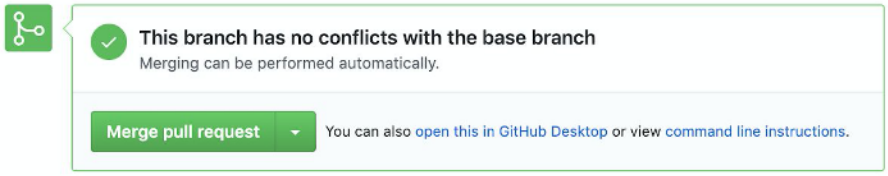


Por otro lado podemos usar la pestaña de “Files changed” para hacer code review.

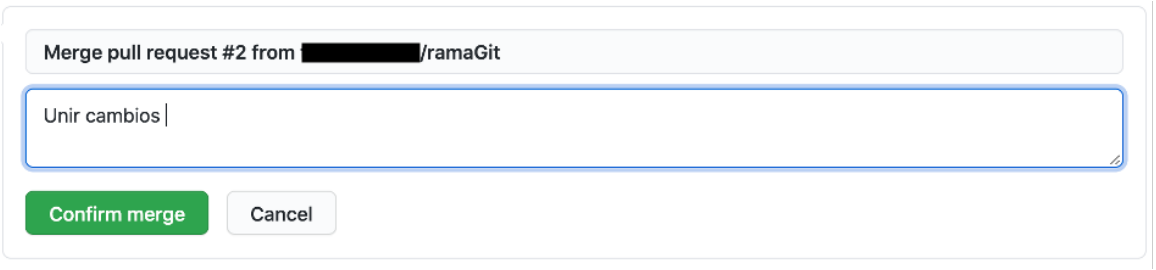


Si detectamos alguna línea de código que requiera cambios puedes clickear sobre ella y agregar un comentario para que el autor del pull request lo modifique. No es necesario volver a crear un nuevo pull request para actualizar los cambios, simplemente haciendo un commit sobre el branch es suficiente,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

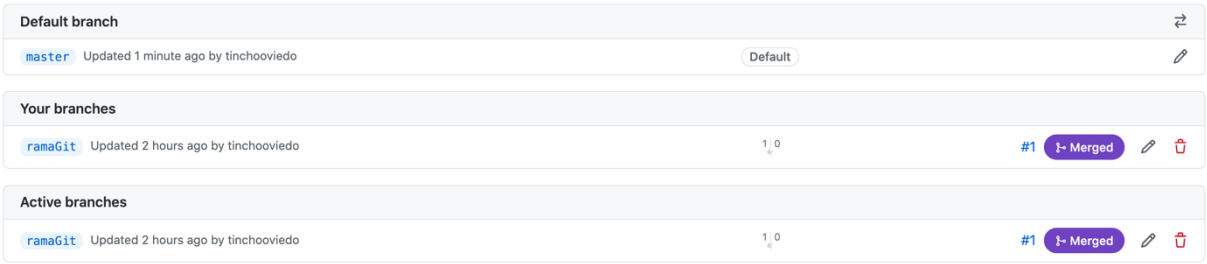
Si esta todo bien y no hay conflictos podemos mergear nuestro branch a master clickeando en el botón “Merge pull request” y de eso modo finaliza el ciclo del branch.



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.



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 pueden 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
```

DESCARGAR UNA RAMA DE REMOTO

A veces ocurre que se generan ramas en remoto, por ejemplo cuando han sido creadas por otros usuarios y subidas al hosting de repositorios, como GitHub o similares, y necesitamos acceder a ellas 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.

El proceso para obtener una rama del repositorio remoto es bien sencillo. Primero usaríamos el comando git checkout para crear la rama que nos falta en local y usamos el -b para pararnos en ella.

```
git checkout -b nombre_de_tu_branch
```

Una vez que hicimos eso, podemos conseguir todo lo que esté en la rama con el comando pull, poniendo el alias del repitorio remoto y el nombre de la rama:

```
git pull origin rama_a_descargar
```

GIT PULL

Acabamos de ver que usamos el comando git pull para descargar la rama, asi que vamos a explicar un poco más de este comando.

Git pull es un comando de Git utilizado para actualizar la versión local de un repositorio desde otro remoto.

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) las nuevas confirmaciones y las fusiona (git merge) en tu rama local.

USANDO GIT PULL

Usa git pull para actualizar un repositorio local del repositorio remoto correspondiente. Por ejemplo: Mientras trabajas localmente en master, ejecuta git pull para actualizar la copia local de master y actualizar las otras ramas remota de seguimiento remoto.

Sin embargo, hay algunas cosas que hay que tener en cuenta para que ese ejemplo sea cierto:

El repositorio local tiene un repositorio remoto vinculado.

- Confirma esto ejecutando git remote -v
- Si existen múltiples remotos, git pull podría no ser suficiente información. Es posible que debas ingresar git pull origin o git pull upstream.

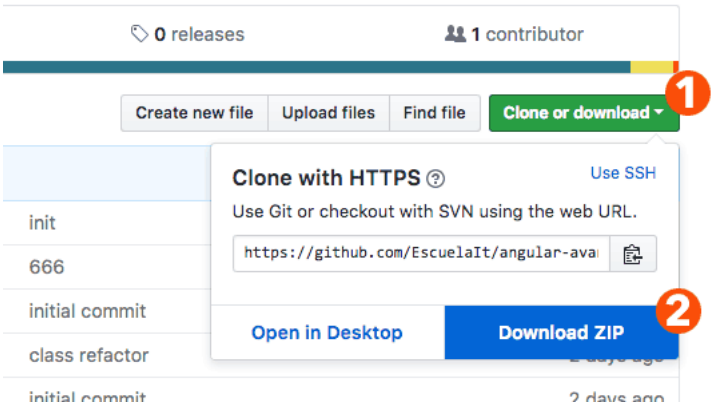
CLONAR UN REPOSITORIO

Ahora vamos a hablar de la operativa de clonado de un repositorio, el proceso que tienes que hacer cuando quieres traerte el código de un proyecto que está publicado en GitHub y lo quieres restaurar en tu ordenador, para poder usarlo en local, modificarlo, etc.

Este paso es bastante básico y muy sencillo de hacer, pero es esencial porque lo necesitarás realizar muchas veces en tu trabajo como desarrollador. Además intentaremos complementarlo con alguna información útil, de modo que puedas aprender cosas útiles y un poquito más avanzadas.

DESCARGAR VS CLONAR

Al inicio de uso de un sitio como GitHub, si no tenemos ni idea de usar Git, también podemos obtener el código de un repositorio descargando un simple Zip. Esta opción la consigues mediante el botón de la siguiente imagen.



Sin embargo, descargar un repositorio así, aunque muy sencillo no te permite algunas de las utilidades interesantes de clonarlo, como:

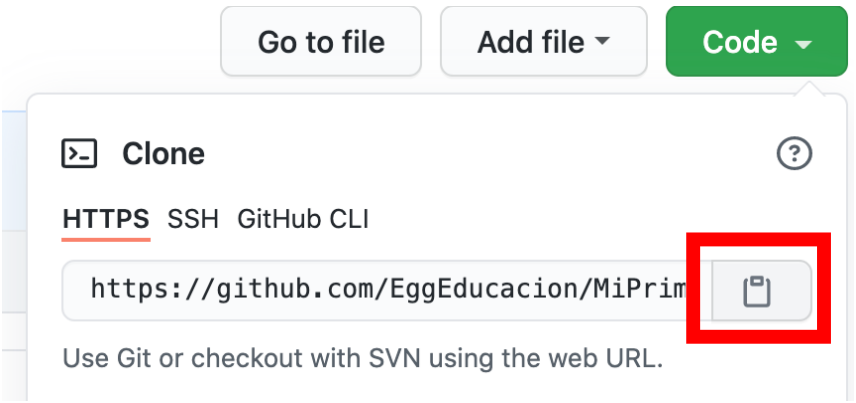
- No crea un repositorio Git en local con los cambios que el repositorio remoto ha tenido a lo largo del tiempo. Es decir, te descargas el código, pero nada más.
- No podrás luego enviar cambios al repositorio remoto, una vez los hayas realizado en local.

En resumen, no podrás usar en general las ventajas de Git en el código descargado. Así que es mejor clonar, ya que aprender a realizar este paso es también muy sencillo.

CLONAR EL REPOSITORIO GIT

Entonces veamos cómo debes clonar el repositorio, de modo que sí puedas beneficiarte de Git con el código descargado. El proceso es el siguiente.

Primero copiarás la URL del repositorio remoto que deseas clonar (ver el icono *"Copy to clipboard"* en la siguiente imagen).



Luego abrirás una ventana de terminal, para situarte sobre la carpeta de tu proyecto que quieras clonar. Yo te recomendaría crear ya directamente una carpeta con el nombre del proyecto que estás clonando, o cualquier otro nombre que te parezca mejor para este repositorio. Te sitúas dentro de esa carpeta y desde ella lanzamos el comando para hacer el clon, que sería algo como esto:

```
git clone https://github.com/EggEducacion/MiPrimerRepositorio.git .
```

El último punto, después de la url copiada desde git, le indica que el clon lo vas a colocar en la carpeta donde estás situado, en tu ventana de terminal. La salida de ese comando sería más o menos como tienes en la siguiente imagen:

```
→  Git git clone https://github.com/EggEducacion/MiPrimerRepositorio.git .
Cloning into '.'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
→  Git git:(master)
```

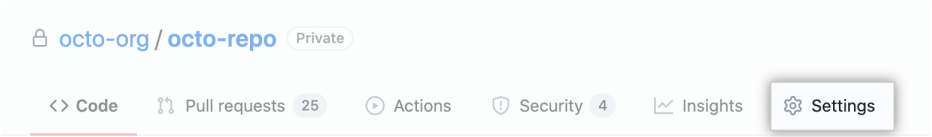
De esta manera nosotros ya tenemos el repositorio remoto para trabajar local y podremos hacer los cambios que queramos y subir los cambios con los comandos que explicamos previamente.

INVITAR COLABORADORES A UN REPOSITORIO PERSONAL

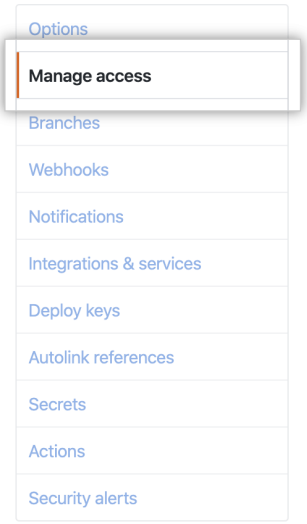
Nosotros vimos como clonar un repositorio para poder usar su código y si quisiéramos hacer cambios y subir esos cambios. Todos los usuarios de GitHub pueden ver y clonar tu repositorio, siempre y cuando sea un repositorio publico. Pero, no todo las personas que clonan tu repositorio pueden subir sus 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:

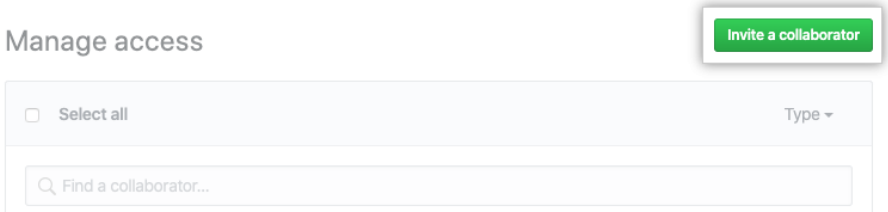
1. Solicita el nombre de usuario de la persona que estás invitando como colaboradora.
2. En GitHub, visita la página principal del repositorio.
3. Debajo de tu nombre de repositorio, da clic en Configuración.



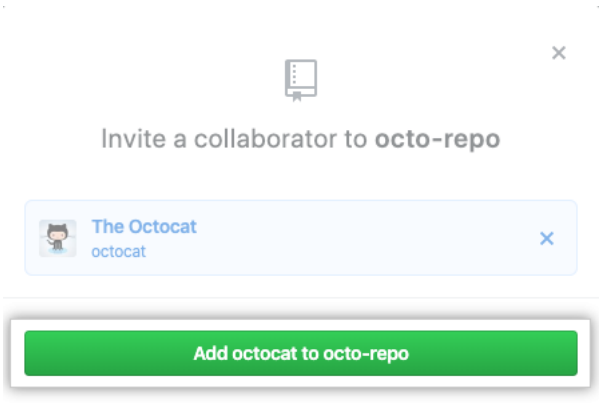
4. En la barra lateral izquierda, da clic en Administrar acceso.



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.

EJERCICIOS DE APRENDIZAJE

Ahora es momento de poner en practica 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 master todos vuestros cambios. Al final les debería quedar un archivo ReadMe con todos sus nombre.

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 master, para que quede el proyecto final.