

CURSO DE PROGRAMACIÓN FULL STACK

TUTORIAL: GUÍA PRÁCTICA DE USO DE GIT CON GITHUB



EGG

GUÍA DE GIT CON GITHUB

¿QUÉ ES EL CONTROL DE VERSIONES?

Los **sistemas de control de versiones** son programas que tienen como objetivo controlar los cambios en el desarrollo de cualquier tipo de *software*, permitiendo conocer el estado actual de un proyecto, los cambios que se le han realizado a cualquiera de sus piezas, las personas que intervinieron en ellos, etc. El *software* de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

Los desarrolladores que trabajan en equipos están escribiendo continuamente nuevo código fuente y cambiando el que ya existe. El código de un proyecto, una aplicación o un componente de *software* normalmente se organiza en una estructura de carpetas o "árbol de archivos". Un desarrollador del equipo podría estar trabajando en una nueva función mientras otro desarrollador soluciona un error no relacionado cambiando código. Cada desarrollador podría hacer sus cambios en varias partes del árbol de archivos.

El control de versiones ayuda a los equipos a resolver estos tipos de problemas, al realizar un *seguimiento* de todos los cambios individuales de cada colaborador y ayudar a evitar que el trabajo concurrente entre en conflicto.

En definitiva, tener un control de los cambios en los códigos de nuestra aplicación es una variable crucial para el éxito de nuestro desarrollo. **Git** es un sistema de control de versiones de código abierto, diseñado para manejar grandes y pequeños proyectos con rapidez y eficiencia. La pretensión de este tutorial es abordar el uso básico de Git proporcionando ejemplos prácticos útiles para comenzar a administrar repositorios remotos con plataformas como Bitbucket o GitHub.

CONTROL DE VERSIONES CON GIT

Git es la mejor opción para la mayoría de los equipos de *software* actuales. Aunque cada equipo es diferente y debería realizar su propio análisis, aquí recogemos los principales motivos por los que destaca el control de versiones de Git con respecto a otras alternativas:

GIT ES UNA EXCELENTE HERRAMIENTA

Git tiene la funcionalidad, el rendimiento, la seguridad y la flexibilidad que la mayoría de los equipos y desarrolladores individuales necesitan. En las comparaciones directas con gran parte de las demás alternativas, Git resulta muy ventajoso para muchos equipos.

GIT ES UN PROYECTO DE CÓDIGO ABIERTO DE CALIDAD

Git es un proyecto de código abierto muy bien respaldado con más de una década de gestión de gran fiabilidad. Los encargados de mantener el proyecto han demostrado un criterio equilibrado y un enfoque maduro para satisfacer las necesidades a largo plazo de sus usuarios con publicaciones periódicas que mejoran la facilidad de uso y la funcionalidad.

La calidad del *software* de código abierto resulta sencilla de analizar y un sin número de empresas dependen en gran medida de esa calidad.

Git goza de una amplia base de usuarios y de un gran apoyo por parte de la comunidad. La documentación es excepcional y para nada escasa, ya que incluye libros, tutoriales y sitios web especializados, así como podcasts y tutoriales en vídeo.

El hecho de que sea de código abierto reduce el costo para los desarrolladores aficionados, puesto que pueden utilizar Git sin necesidad de pagar ninguna cuota. En lo que respecta a los proyectos de código abierto, no cabe duda de que Git es el sucesor de las anteriores generaciones de los exitosos sistemas de control de versiones de código abierto, SVN y CVS.

GITHUB

Github es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador. La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas, y que como usuario no solo puedas descargar la aplicación, sino también entrar a su perfil para leer sobre ella o colaborar con su desarrollo.

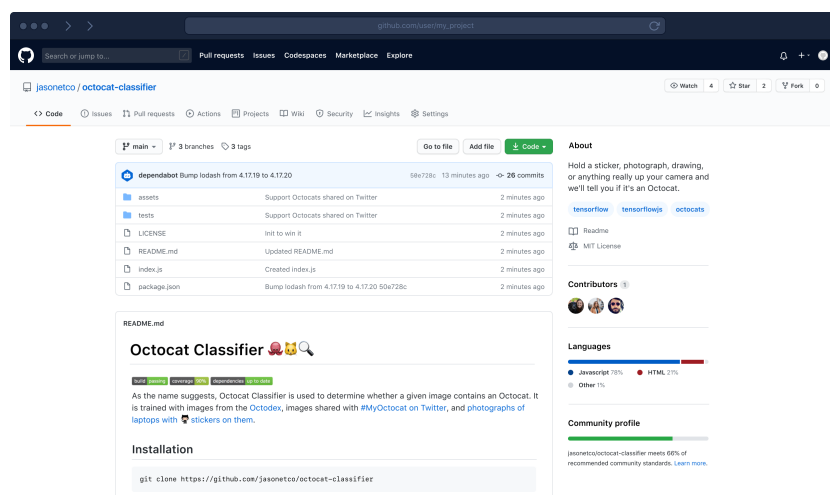
Un sistema de gestión de versiones es ese con el que **los desarrolladores pueden administrar su proyecto**, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

Git, al ser un sistema de control, va a ser la herramienta que nos va a permitir comparar el código de un archivo **para ver las diferencias entre las versiones**, restaurar versiones antiguas si algo sale mal, y fusionar los cambios de distintas versiones.

Así pues, Github es un portal para gestionar proyectos usando el sistema Git.

¿CÓMO VAMOS A LOGRAR ESO?

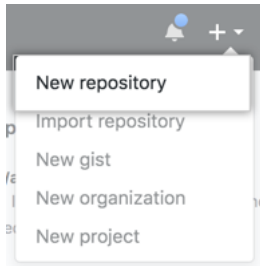
Github permite que los desarrolladores alojen proyectos creando repositorios de forma gratuita. Un repositorio es como una carpeta para tu proyecto. El repositorio de tu proyecto contiene todos los archivos de tu repositorio y almacena el historial de revisión de cada archivo. También puedes debatir y administrar el trabajo de tu proyecto dentro del repositorio.



CREAR UN REPOSITORIO EN GITHUB

Para subir tu proyecto a GitHub, deberás crear un repositorio donde alojarlo. Para poder crear un repositorio deberemos crearnos una cuenta en GitHub.

- 1) En la esquina superior derecha de cualquier página, utiliza el menú desplegable **+** y selecciona **Repositorio Nuevo**



- 2) Escribe un nombre corto y fácil de recordar para tu repositorio. Por ejemplo: "hola-mundo".

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

- 3) También puedes agregar una descripción de tu repositorio. Por ejemplo, "Mi primer repositorio en GitHub".

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

My first repository on GitHub

- 4) Elige la visibilidad del repositorio. Puedes restringir quién tiene acceso a un repositorio eligiendo la visibilidad de un repositorio: público o privado. Público significa que cualquier persona puede ver ese repositorio y privado significa que solo personas autorizadas pueden verlo. Que sea público no significa que la gente puede subir cosas a nuestro repositorio, lo único que permite es que se puedan ver los archivos.

Description (optional)

- ☒ **Public**
Anyone can see this repository. You choose who can commit.
- ☐ **Internal**
Octo Corp **enterprise members** can see this repository. You choose who can commit.
- ☐ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

5) Podemos crear el repositorio con un ReadMe

- ☒ **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.

Skip this step if you're importing an existing repository.

- ☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

6) Da click en Crear repositorio.

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

INSTALACIÓN DE GIT

Una vez que tenemos creado un repositorio en GitHub vamos a tener que instalar Git

INSTALADOR DE GIT PARA MAC

El modelo de Git de Apple viene preinstalado en macOS. Abra su terminal o editor de scripts de shell de su elección e **ingrese git --version** para verificar qué versión de Git está en su máquina. Si aún no está en su máquina, ejecutar **git --version** le pedirá que instale Git.

Si bien esta compilación de Git está bien para algunos usuarios, es posible que desee instalar la versión más actualizada. Puede hacerlo de muchas formas diferentes; hemos recopilado algunas de las opciones más fáciles a continuación.

1) Una forma es instalar Git en un Mac es mediante el instalador independiente:

A. Descarga el instalador de Git para Mac más reciente desde acá:

<https://sourceforge.net/projects/git-osx-installer/files/>

B. Sigue las instrucciones para instalar Git.

Abre un terminal y escribe el siguiente texto para comprobar que la instalación se ha realizado correctamente:

```
git --version:
```

```
$ git --version  
git version 2.9.2
```

2) Otra forma es instalar Git mediante HomeBrew:

Homebrew instala una lista de paquetes útiles que no vienen preinstalados en Mac.

A. Pegue el siguiente comando en su terminal para instalar Homebrew:

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install  
.sh)"
```

B. El terminal le pedirá que ingrese una contraseña. Ingrese la contraseña que usa para iniciar sesión en su Mac y continuar con el proceso de instalación.

C. Una vez terminado, ingrese **brew install git** en la terminal y espere a que se descargue. Verifique que Git se instaló ejecutando **git --version**.

INSTALADOR DE GIT PARA LINUX

Debian / Ubuntu (apt-get)

Los paquetes de Git están disponibles mediante APT:

1. Desde tu núcleo, instala Git mediante apt-get:

```
$ sudo apt-get update  
$ sudo apt-get install git
```

2. Introduce el siguiente texto para verificar que la instalación se ha realizado correctamente:

```
git --version:
```

```
$ git --version  
git version 2.9.2
```

INSTALADOR DE GIT PARA WINDOWS

1. Descarga el [instalador de Git para Windows](#) más reciente.
2. Cuando hayas iniciado correctamente el instalador, verás la pantalla del asistente de instalación de Git.
3. Selecciona las opciones Siguiente y Finalizar para completar la instalación. Las opciones predeterminadas son las más lógicas en la mayoría de los casos.

4. Abre el símbolo del sistema (o Git Bash si durante la instalación seleccionaste no usar Git desde el símbolo del sistema de Windows).
5. Introduce el siguiente texto para verificar que la instalación se ha realizado correctamente:

```
git --version:
```

```
$ git --version
git version 2.9.2
```

CONFIGURACIÓN INICIAL

Abra su terminal de Git para comenzar con la ejecución de comandos, por ejemplo, abrirá el programa **Git bash** en Windows para ingresar a la línea de comandos de este programa.

Una vez que ingrese, use el siguiente comando para establecer el nombre de usuario de git:

```
git config --global user.name "Jhoel Perez"
```

Recuerde sustituir el texto entre comillas por su nombre real. Ahora indique el correo electrónico del usuario para git:

```
git config --global user.email "micorreopersonal@jhoel.com"
```

Sustituyendo el texto entre comillas por su cuenta de correo electrónico. Esta configuración inicial debería ser suficiente para comenzar. Para comprobar otros valores de su configuración actual ejecute:

```
git config --list
```

Se mostrarán los nuevos valores configurados al final, y otros valores de configuración predeterminados:

```
...
color.diff=auto
color.status=auto
...
user.name=Juan Perez
user.email=micorreopersonal@juan.com
```

FORMAS DE COMENZAR A TRABAJAR CON GIT

Para trabajar con Git con Github tenemos dos formas de trabajar:

- 1) Trabajar en local, en un repositorio que me cree en mi máquina y vincularlo a un repositorio creado en GitHub
- 2) Clonar un repositorio de Github (u otro hosting de repositorios) para traernos a local el repositorio completo y empezar a trabajar con ese proyecto.

Vamos a elegir de momento la opción 1) que nos permitirá comenzar desde cero y con la que podremos apreciar mejor cuáles son las operaciones básicas con Git. En este sentido, cualquier operación que realizas con Git tiene que comenzar mediante el trabajo en local, por lo que tienes que comenzar por crear el repositorio en tu propia máquina. Incluso si tus objetivos son simplemente subir ese repositorio a Github para que otras personas lo puedan acceder a través del hosting remoto de repositorios, tienes que comenzar trabajando en local.

INICIO DE UN NUEVO REPOSITORIO: GIT INIT

Primero deberemos crear una carpeta vacía para inicializar nuestro repositorio, una vez que la tenemos creada debemos decirle a la terminal que se pare en esa carpeta, así todos los comandos de Git, afectan a esa carpeta.

En Windows podemos hacer click derecho a la carpeta y darle a **Git Bash Here**, eso nos abrirá la terminal Git Bash en la carpeta para trabajar Git.

En Mac podemos hacer lo mismo, click derecho a la carpeta y darle a la opción **Nuevo Terminal en la carpeta**.

Una vez parados en nuestra carpeta. Para crear un nuevo repositorio, usa el comando **git init**. `git init` es un comando que se utiliza una sola vez durante la configuración inicial de un repositorio nuevo. Al ejecutar este comando, se creará un nuevo subdirectorio **.git** en tu directorio de trabajo actual. También se creará una nueva rama principal.

```
$ git init
```

GUARDAR CAMBIOS EN EL REPOSITORIO: GIT ADD Y GIT COMMIT

Ahora que has iniciado o clonado un repositorio, puedes realizar commits en la versión del archivo. Vamos a ir a nuestra carpeta y vamos a crear un archivo txt con nuestro nombre escrito, una vez que lo tengamos creado vamos a hacer los siguientes comandos.

Git Status

El comando de `git status` nos da toda la información necesaria sobre la rama actual.

```
git status
```

Podemos encontrar información como:

- Si la rama actual está actualizada
- Si hay algo para confirmar, enviar o recibir (pull).
- Si hay archivos en preparación (staged), sin preparación(unstaged) o que no están recibiendo seguimiento (untracked)
- Si hay archivos creados, modificados o eliminados


```
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory
)

    modified:   src/App.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    src/components/
```

Nota: veremos el concepto de ramas más adelante.

Hacemos un git status en nuestro repositorio para ver el estado actual.

Git Add

Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

Necesitamos usar el comando git add para incluir los cambios del o de los archivos en tu siguiente commit.

Añadir un único archivo:

```
git add <archivo>
```

Añadir todo de una vez:

```
git add .
```

Si revisas la captura de pantalla del git status, verás que hay nombres de archivos en rojo - esto significa que los archivos sin preparación. Estos archivos no serán incluidos en tus commits hasta que no los añadas.

Hacemos un **git add .** para agregar nuestro archivo txt. Una vez que hacemos el git add hacemos otro git status, ahora veremos que los archivos que estaban en rojo, están en verde, esto quiere decir que ya los hemos agregado para hacer nuestro commit.

```
Cem-MacBook-Pro:my-new-app cem$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   src/App.js
    new file:   src/components/myFirstComponent.js
```

Git Commit

Este sea quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios (quizás después de una tarea o asunto específico) o subir un archivo / proyecto.

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.

También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente.

```
git commit -m "mensaje de confirmación"
```

Hacemos un commit para guardar nuestro txt y le ponemos un mensaje que explique que hemos hecho.

VINCULAR NUESTRO REPOSITORIO CON GITHUB

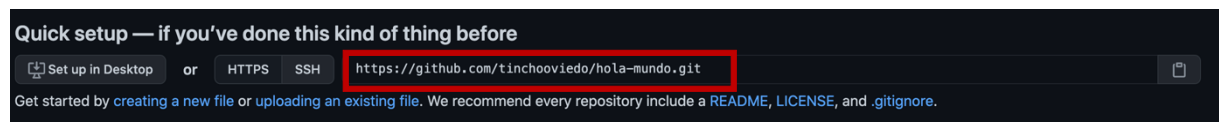
Primero deberemos crear un repositorio, **sin el archivo readME**.

Una vez que tenemos el archivo agregado y guardado de manera local, tenemos que vincular este repositorio local a un repositorio remoto en GitHub. Para esto vamos a utilizar el comando **git remote add**.

Este comando va a tomar el alias nuestro repositorio y la url de nuestro repositorio en GitHub, con esto va a vincularlo con nuestro repositorio local.

```
git remote add <name> <url>
```

El alias que vamos a utilizar para Github es **origin** y para obtener la url de nuestro repositorio, podemos encontrarla al principio de nuestro repositorio:



Buscamos el url de nuestro repositorio de GitHub y lo hacemos en nuestro terminal.

Por lo que pondremos `git remote add origin <url>`

COLABORACIÓN ENTRE REPOSITORIOS: GIT PUSH

Después de haber confirmado tus cambios y haber vinculado el repositorio de Git local con el repositorio remoto de GitHub, el siguiente paso que quieres dar es enviar tus cambios o archivos al servidor remoto. **git push** envía tus commits al repositorio remoto.

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

De todas formas, si tu rama ha sido creada recientemente, puede que tengas que cargar y subir tu rama con el siguiente comando:

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

Recordemos que usamos el alias origin para el repositorio remoto.

Cuando creamos un repositorio en GitHub, nos crea una rama por defecto llamada main, podemos en la configuración cambiar para que la rama que se cree por defecto se llame master.

Por lo que ahora haremos el siguiente comando en nuestra terminal:

```
git push origin main
```

o

```
git push origin master
```

Una vez que hemos hecho esto, si refrescamos nuestro repositorio vamos a ver nuestro archivo txt en nuestro repositorio de GitHub.

Este proceso se va a repetir, quitando la vinculación y la inicialización del repositorio, cada vez que nosotros hagamos un cambio dentro de nuestro repositorio. Esto puede ser modificar un archivo ya existente o agregar más archivos a la carpeta local.

RAMAS EN GIT

A menudo necesitamos trabajar con más de una persona sobre un mismo proyecto. Pero, ¿Qué pasa si más de un desarrollador hace cambios sobre el mismo archivo?, o peor aún, ¿Qué pasa si ambos cambian la misma línea de código?

Para evitar este tipo de problemas y colisionar código permanentemente, git provee la herramienta de branches (rama). De esta manera, puedes crear tu 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.

GIT CHECKOUT Y GIT BRANCH

El comando **git branch** 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

Puedes comenzar tu primera práctica para trabajar con ramas. Haremos algo tan sencillo como lanzar el comando *"git branch"* a secas. 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.

LA RAMA MASTER O MAIN

Cuando inicializamos un proyecto con Git automáticamente nos encontramos en una rama a la que se denomina "master".

Puedes ver la rama en la que te encuentras en cada instante con el comando:

```
git branch
```

Esta rama es la principal de tu proyecto y a partir de la que podrás crear nuevas ramas cuando lo necesites.

Si has hecho algún commit en tu repositorio observarás que después de lanzar el comando *"git branch"* nos informa el nombre de la rama como "master".

Recordemos que en GitHub esta rama puede llamarse Main, siempre podemos cambiar el nombre de la rama a Master con las configuraciones de GitHub.

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"
```

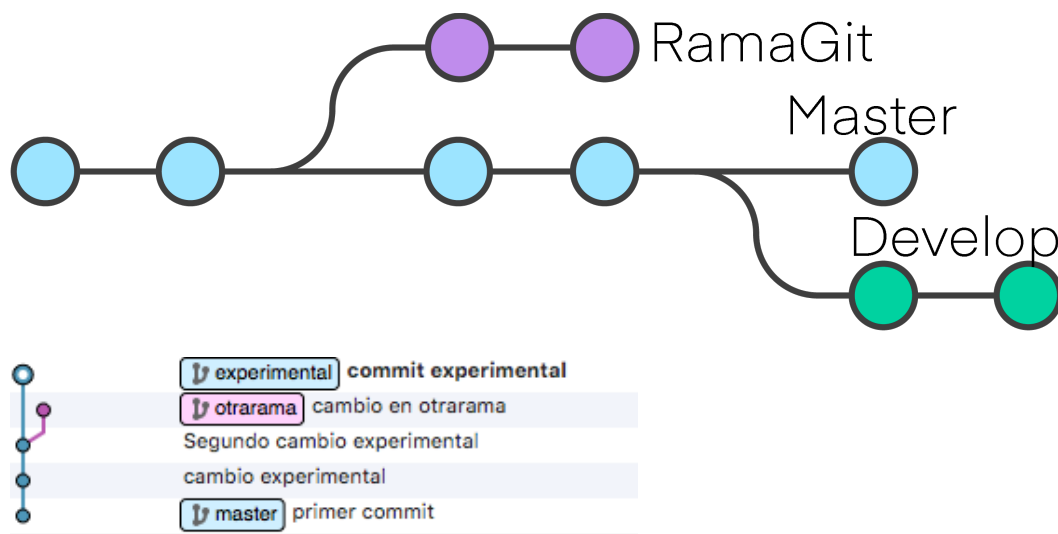
No vamos a hacer un push todavía porque eso lo vamos a explicar más adelante, ya que eso hará que nuestra rama aparezca en el repositorio remoto.

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.

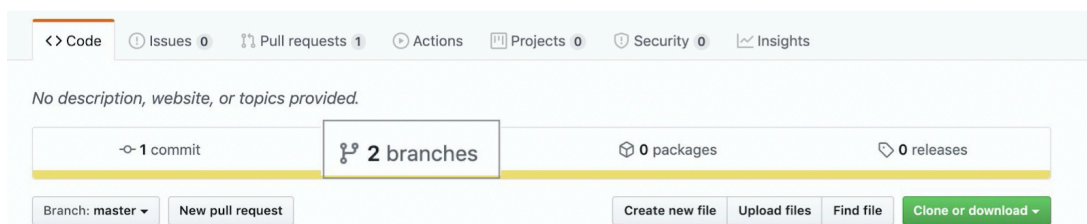
Esto sería así:

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

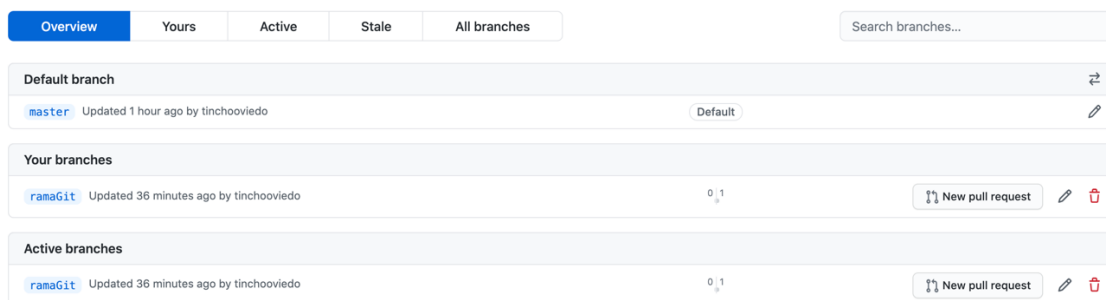
Una vez esto hecho esto podríamos pararnos en nuestra rama y simplemente escribir:

```
git push
```

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



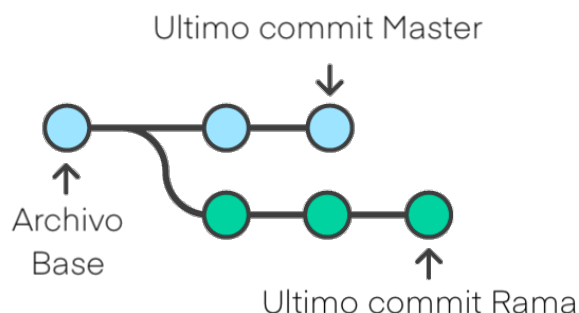
Y ahí podremos ver las dos ramas

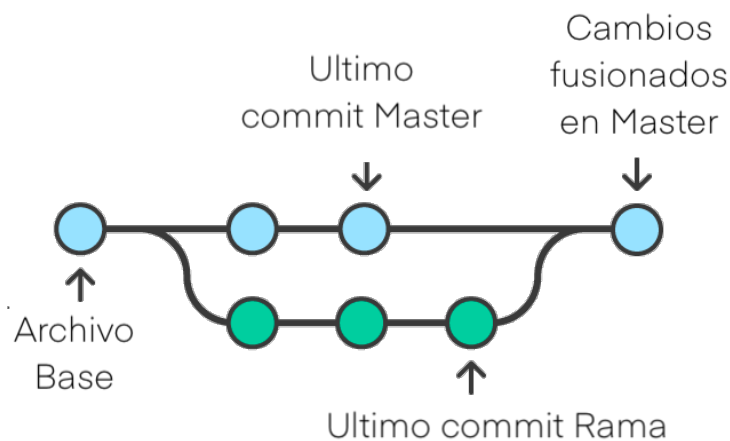


Puedes subir tanto commits creas convenientes a tu branch antes de mergear a master, siempre es mejor pequeños y frecuentes cambios que pocos y grandes.

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 fusión 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 clicar en el botón **New pull request**.

Overview | Yours | Active | Stale | All branches

Search branches...

Default branch

master Updated 2 hours ago by gonzalogEB **Default**

Your branches

another_random_branch Updated 2 hours ago by gonzalogEB 0|0 **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 así se incrementa el número según la cantidad de commits que este por delante master. Ahora, si el número estuviera a la derecha, sería que la otra rama está x commits por delante master.

Aca podemos ver un ejemplo con ramaGit:

Your branches

ramaGit Updated 1 hour ago by tinchooviedo 1 commit ahead of master **New pull request**

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.

GonzaGr92 / node_server Private

Watch 1 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Actions Projects 0 Security 0 Insights

Open a pull request

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

base: master ← compare: another_random_branch **1** Able to merge. These branches can be automatically merged.

2 **Migrate to sequelize**

Write | Preview | H | B | I | | | | | | | | |

Leave a comment

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

Create pull request

3 **Reviewers**
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked issues
Use **Closing keywords** in the description to automatically close issues

Helpful resources
GitHub Community Guidelines

4 **Showing 4 changed files with 158 additions and 11 deletions.**

Unified | Split

25 app.js

```
@@ -7,6 +7,9 @@ const _ = require('lodash');
7 7   const QUERY_ALL = 'ALL';
8 8   const QUERY_GET = 'GET';
```


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 título 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.

The screenshot displays a GitHub Pull Request interface. At the top, navigation tabs include Code, Issues (0), Pull requests (1), Actions, Projects (0), Security (0), and Insights. The title of the pull request is "Migrate to sequelize #3". Below the title, it states "gonzalogEB wants to merge 1 commit into master from another_random_branch". A green "Open" button is visible. The main content area shows a conversation with a comment from "gonzalogEB" saying "Este es un comentario". Below the comment is a commit titled "Migrate to sequelize" with the hash 78f236f. A green box with a checkmark indicates "This branch has no conflicts with the base branch" and "Merging can be performed automatically." Below this box is a "Merge pull request" button. The right sidebar contains sections for Reviewers, Assignees, Labels, Projects, Milestone, Linked issues, and Notifications. At the bottom, there are buttons for "Close pull request" and "Comment".

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

Migrate to sequelize #3

gonzalogEB wants to merge 1 commit into master from another_random_branch

Changes from all commits • File filter... • Jump to... • 0 / 4 files viewed • Review changes

25 app.js

```
@@ -7,6 +7,9 @@ const _ = require('lodash');
7   const QUERY_ALL = 'ALL';
8   const QUERY_GET = 'GET';
9
10  // Connect to sqlite db
11  let db = new sqlite3.Database('./db/chinook.db', (err) => {
12    if (err) {
13      // Project self dependencies
14      + var Artist = require('./models/Artist');
15      // Connect to sqlite db
16      let db = new sqlite3.Database('./db/chinook.db', (err) => {
17        if (err) {
18          const QUERY_ALL = 'ALL';
19          const QUERY_GET = 'GET';
20
21          // Project self dependencies
22          + var Artist = require('./models/Artist');
23          // Connect to sqlite db
24          let db = new sqlite3.Database('./db/chinook.db', (err) => {
25            if (err) {
26              const page = _.get(req, 'query.page', 0);
27              const limit = 20;
28
29              + Artist.findAll({
30                + where: {
31                  + ArtistId: 2
32                }
33              });
34
35              let sql = `SELECT * FROM artists
36                LIMIT ${limit}
37                OFFSET ${limit * page}`;
38
39              const page = _.get(req, 'query.page', 0);
40              const limit = 20;
41
42              + Artist.findAll({
43                + where: {
44                  + ArtistId: 2
45                }
46              });
47
48              let sql = `SELECT * FROM artists
49                LIMIT ${limit}
50                OFFSET ${limit * page}`;
51
52              const page = _.get(req, 'query.page', 0);
53              const limit = 20;
54
55              + Artist.findAll({
56                + where: {
57                  + ArtistId: 2
58                }
59              });
60
61              let sql = `SELECT * FROM artists
62                LIMIT ${limit}
63                OFFSET ${limit * page}`;
64
65              const page = _.get(req, 'query.page', 0);
66              const limit = 20;
67
68              + Artist.findAll({
69                + where: {
70                  + ArtistId: 2
71                }
72              });
73
74              let sql = `SELECT * FROM artists
75                LIMIT ${limit}
76                OFFSET ${limit * page}`;
77
78              const page = _.get(req, 'query.page', 0);
79              const limit = 20;
80
81              + Artist.findAll({
82                + where: {
83                  + ArtistId: 2
84                }
85              });
86
87              let sql = `SELECT * FROM artists
88                LIMIT ${limit}
89                OFFSET ${limit * page}`;
90
91              const page = _.get(req, 'query.page', 0);
92              const limit = 20;
93
94              + Artist.findAll({
95                + where: {
96                  + ArtistId: 2
97                }
98              });
99
100             let sql = `SELECT * FROM artists
101               LIMIT ${limit}
102               OFFSET ${limit * page}`;
103
104             const page = _.get(req, 'query.page', 0);
105             const limit = 20;
106
107             + Artist.findAll({
108               + where: {
109                 + ArtistId: 2
110               }
111             });
112
113             let sql = `SELECT * FROM artists
114               LIMIT ${limit}
115               OFFSET ${limit * page}`;
116
117             const page = _.get(req, 'query.page', 0);
118             const limit = 20;
119
120             + Artist.findAll({
121               + where: {
122                 + ArtistId: 2
123               }
124             });
125
126             let sql = `SELECT * FROM artists
127               LIMIT ${limit}
128               OFFSET ${limit * page}`;
129
130             const page = _.get(req, 'query.page', 0);
131             const limit = 20;
132
133             + Artist.findAll({
134               + where: {
135                 + ArtistId: 2
136               }
137             });
138
139             let sql = `SELECT * FROM artists
140               LIMIT ${limit}
141               OFFSET ${limit * page}`;
142
143             const page = _.get(req, 'query.page', 0);
144             const limit = 20;
145
146             + Artist.findAll({
147               + where: {
148                 + ArtistId: 2
149               }
150             });
151
152             let sql = `SELECT * FROM artists
153               LIMIT ${limit}
154               OFFSET ${limit * page}`;
155
156             const page = _.get(req, 'query.page', 0);
157             const limit = 20;
158
159             + Artist.findAll({
160               + where: {
161                 + ArtistId: 2
162               }
163             });
164
165             let sql = `SELECT * FROM artists
166               LIMIT ${limit}
167               OFFSET ${limit * page}`;
168
169             const page = _.get(req, 'query.page', 0);
170             const limit = 20;
171
172             + Artist.findAll({
173               + where: {
174                 + ArtistId: 2
175               }
176             });
177
178             let sql = `SELECT * FROM artists
179               LIMIT ${limit}
180               OFFSET ${limit * page}`;
181
182             const page = _.get(req, 'query.page', 0);
183             const limit = 20;
184
185             + Artist.findAll({
186               + where: {
187                 + ArtistId: 2
188               }
189             });
190
191             let sql = `SELECT * FROM artists
192               LIMIT ${limit}
193               OFFSET ${limit * page}`;
194
195             const page = _.get(req, 'query.page', 0);
196             const limit = 20;
197
198             + Artist.findAll({
199               + where: {
200                 + ArtistId: 2
201               }
202             });
203
204             let sql = `SELECT * FROM artists
205               LIMIT ${limit}
206               OFFSET ${limit * page}`;
207
208             const page = _.get(req, 'query.page', 0);
209             const limit = 20;
210
211             + Artist.findAll({
212               + where: {
213                 + ArtistId: 2
214               }
215             });
216
217             let sql = `SELECT * FROM artists
218               LIMIT ${limit}
219               OFFSET ${limit * page}`;
220
221             const page = _.get(req, 'query.page', 0);
222             const limit = 20;
223
224             + Artist.findAll({
225               + where: {
226                 + ArtistId: 2
227               }
228             });
229
230             let sql = `SELECT * FROM artists
231               LIMIT ${limit}
232               OFFSET ${limit * page}`;
233
234             const page = _.get(req, 'query.page', 0);
235             const limit = 20;
236
237             + Artist.findAll({
238               + where: {
239                 + ArtistId: 2
240               }
241             });
242
243             let sql = `SELECT * FROM artists
244               LIMIT ${limit}
245               OFFSET ${limit * page}`;
246
247             const page = _.get(req, 'query.page', 0);
248             const limit = 20;
249
250             + Artist.findAll({
251               + where: {
252                 + ArtistId: 2
253               }
254             });
255
256             let sql = `SELECT * FROM artists
257               LIMIT ${limit}
258               OFFSET ${limit * page}`;
259
260             const page = _.get(req, 'query.page', 0);
261             const limit = 20;
262
263             + Artist.findAll({
264               + where: {
265                 + ArtistId: 2
266               }
267             });
268
269             let sql = `SELECT * FROM artists
270               LIMIT ${limit}
271               OFFSET ${limit * page}`;
272
273             const page = _.get(req, 'query.page', 0);
274             const limit = 20;
275
276             + Artist.findAll({
277               + where: {
278                 + ArtistId: 2
279               }
280             });
281
282             let sql = `SELECT * FROM artists
283               LIMIT ${limit}
284               OFFSET ${limit * page}`;
285
286             const page = _.get(req, 'query.page', 0);
287             const limit = 20;
288
289             + Artist.findAll({
290               + where: {
291                 + ArtistId: 2
292               }
293             });
294
295             let sql = `SELECT * FROM artists
296               LIMIT ${limit}
297               OFFSET ${limit * page}`;
298
299             const page = _.get(req, 'query.page', 0);
300             const limit = 20;
301
302             + Artist.findAll({
303               + where: {
304                 + ArtistId: 2
305               }
306             });
307
308             let sql = `SELECT * FROM artists
309               LIMIT ${limit}
310               OFFSET ${limit * page}`;
311
312             const page = _.get(req, 'query.page', 0);
313             const limit = 20;
314
315             + Artist.findAll({
316               + where: {
317                 + ArtistId: 2
318               }
319             });
320
321             let sql = `SELECT * FROM artists
322               LIMIT ${limit}
323               OFFSET ${limit * page}`;
324
325             const page = _.get(req, 'query.page', 0);
326             const limit = 20;
327
328             + Artist.findAll({
329               + where: {
330                 + ArtistId: 2
331               }
332             });
333
334             let sql = `SELECT * FROM artists
335               LIMIT ${limit}
336               OFFSET ${limit * page}`;
337
338             const page = _.get(req, 'query.page', 0);
339             const limit = 20;
340
341             + Artist.findAll({
342               + where: {
343                 + ArtistId: 2
344               }
345             });
346
347             let sql = `SELECT * FROM artists
348               LIMIT ${limit}
349               OFFSET ${limit * page}`;
350
351             const page = _.get(req, 'query.page', 0);
352             const limit = 20;
353
354             + Artist.findAll({
355               + where: {
356                 + ArtistId: 2
357               }
358             });
359
360             let sql = `SELECT * FROM artists
361               LIMIT ${limit}
362               OFFSET ${limit * page}`;
363
364             const page = _.get(req, 'query.page', 0);
365             const limit = 20;
366
367             + Artist.findAll({
368               + where: {
369                 + ArtistId: 2
370               }
371             });
372
373             let sql = `SELECT * FROM artists
374               LIMIT ${limit}
375               OFFSET ${limit * page}`;
376
377             const page = _.get(req, 'query.page', 0);
378             const limit = 20;
379
380             + Artist.findAll({
381               + where: {
382                 + ArtistId: 2
383               }
384             });
385
386             let sql = `SELECT * FROM artists
387               LIMIT ${limit}
388               OFFSET ${limit * page}`;
389
390             const page = _.get(req, 'query.page', 0);
391             const limit = 20;
392
393             + Artist.findAll({
394               + where: {
395                 + ArtistId: 2
396               }
397             });
398
399             let sql = `SELECT * FROM artists
400               LIMIT ${limit}
401               OFFSET ${limit * page}`;
402
403             const page = _.get(req, 'query.page', 0);
404             const limit = 20;
405
406             + Artist.findAll({
407               + where: {
408                 + ArtistId: 2
409               }
410             });
411
412             let sql = `SELECT * FROM artists
413               LIMIT ${limit}
414               OFFSET ${limit * page}`;
415
416             const page = _.get(req, 'query.page', 0);
417             const limit = 20;
418
419             + Artist.findAll({
420               + where: {
421                 + ArtistId: 2
422               }
423             });
424
425             let sql = `SELECT * FROM artists
426               LIMIT ${limit}
427               OFFSET ${limit * page}`;
428
429             const page = _.get(req, 'query.page', 0);
430             const limit = 20;
431
432             + Artist.findAll({
433               + where: {
434                 + ArtistId: 2
435               }
436             });
437
438             let sql = `SELECT * FROM artists
439               LIMIT ${limit}
440               OFFSET ${limit * page}`;
441
442             const page = _.get(req, 'query.page', 0);
443             const limit = 20;
444
445             + Artist.findAll({
446               + where: {
447                 + ArtistId: 2
448               }
449             });
450
451             let sql = `SELECT * FROM artists
452               LIMIT ${limit}
453               OFFSET ${limit * page}`;
454
455             const page = _.get(req, 'query.page', 0);
456             const limit = 20;
457
458             + Artist.findAll({
459               + where: {
460                 + ArtistId: 2
461               }
462             });
463
464             let sql = `SELECT * FROM artists
465               LIMIT ${limit}
466               OFFSET ${limit * page}`;
467
468             const page = _.get(req, 'query.page', 0);
469             const limit = 20;
470
471             + Artist.findAll({
472               + where: {
473                 + ArtistId: 2
474               }
475             });
476
477             let sql = `SELECT * FROM artists
478               LIMIT ${limit}
479               OFFSET ${limit * page}`;
480
481             const page = _.get(req, 'query.page', 0);
482             const limit = 20;
483
484             + Artist.findAll({
485               + where: {
486                 + ArtistId: 2
487               }
488             });
489
490             let sql = `SELECT * FROM artists
491               LIMIT ${limit}
492               OFFSET ${limit * page}`;
493
494             const page = _.get(req, 'query.page', 0);
495             const limit = 20;
496
497             + Artist.findAll({
498               + where: {
499                 + ArtistId: 2
500               }
501             });
502
503             let sql = `SELECT * FROM artists
504               LIMIT ${limit}
505               OFFSET ${limit * page}`;
506
507             const page = _.get(req, 'query.page', 0);
508             const limit = 20;
509
510             + Artist.findAll({
511               + where: {
512                 + ArtistId: 2
513               }
514             });
515
516             let sql = `SELECT * FROM artists
517               LIMIT ${limit}
518               OFFSET ${limit * page}`;
519
520             const page = _.get(req, 'query.page', 0);
521             const limit = 20;
522
523             + Artist.findAll({
524               + where: {
525                 + ArtistId: 2
526               }
527             });
528
529             let sql = `SELECT * FROM artists
530               LIMIT ${limit}
531               OFFSET ${limit * page}`;
532
533             const page = _.get(req, 'query.page', 0);
534             const limit = 20;
535
536             + Artist.findAll({
537               + where: {
538                 + ArtistId: 2
539               }
540             });
541
542             let sql = `SELECT * FROM artists
543               LIMIT ${limit}
544               OFFSET ${limit * page}`;
545
546             const page = _.get(req, 'query.page', 0);
547             const limit = 20;
548
549             + Artist.findAll({
550               + where: {
551                 + ArtistId: 2
552               }
553             });
554
555             let sql = `SELECT * FROM artists
556               LIMIT ${limit}
557               OFFSET ${limit * page}`;
558
559             const page = _.get(req, 'query.page', 0);
560             const limit = 20;
561
562             + Artist.findAll({
563               + where: {
564                 + ArtistId: 2
565               }
566             });
567
568             let sql = `SELECT * FROM artists
569               LIMIT ${limit}
570               OFFSET ${limit * page}`;
571
572             const page = _.get(req, 'query.page', 0);
573             const limit = 20;
574
575             + Artist.findAll({
576               + where: {
577                 + ArtistId: 2
578               }
579             });
580
581             let sql = `SELECT * FROM artists
582               LIMIT ${limit}
583               OFFSET ${limit * page}`;
584
585             const page = _.get(req, 'query.page', 0);
586             const limit = 20;
587
588             + Artist.findAll({
589               + where: {
590                 + ArtistId: 2
591               }
592             });
593
594             let sql = `SELECT * FROM artists
595               LIMIT ${limit}
596               OFFSET ${limit * page}`;
597
598             const page = _.get(req, 'query.page', 0);
599             const limit = 20;
600
601             + Artist.findAll({
602               + where: {
603                 + ArtistId: 2
604               }
605             });
606
607             let sql = `SELECT * FROM artists
608               LIMIT ${limit}
609               OFFSET ${limit * page}`;
610
611             const page = _.get(req, 'query.page', 0);
612             const limit = 20;
613
614             + Artist.findAll({
615               + where: {
616                 + ArtistId: 2
617               }
618             });
619
620             let sql = `SELECT * FROM artists
621               LIMIT ${limit}
622               OFFSET ${limit * page}`;
623
624             const page = _.get(req, 'query.page', 0);
625             const limit = 20;
626
627             + Artist.findAll({
628               + where: {
629                 + ArtistId: 2
630               }
631             });
632
633             let sql = `SELECT * FROM artists
634               LIMIT ${limit}
635               OFFSET ${limit * page}`;
636
637             const page = _.get(req, 'query.page', 0);
638             const limit = 20;
639
640             + Artist.findAll({
641               + where: {
642                 + ArtistId: 2
643               }
644             });
645
646             let sql = `SELECT * FROM artists
647               LIMIT ${limit}
648               OFFSET ${limit * page}`;
649
650             const page = _.get(req, 'query.page', 0);
651             const limit = 20;
652
653             + Artist.findAll({
654               + where: {
655                 + ArtistId: 2
656               }
657             });
658
659             let sql = `SELECT * FROM artists
660               LIMIT ${limit}
661               OFFSET ${limit * page}`;
662
663             const page = _.get(req, 'query.page', 0);
664             const limit = 20;
665
666             + Artist.findAll({
667               + where: {
668                 + ArtistId: 2
669               }
670             });
671
672             let sql = `SELECT * FROM artists
673               LIMIT ${limit}
674               OFFSET ${limit * page}`;
675
676             const page = _.get(req, 'query.page', 0);
677             const limit = 20;
678
679             + Artist.findAll({
680               + where: {
681                 + ArtistId: 2
682               }
683             });
684
685             let sql = `SELECT * FROM artists
686               LIMIT ${limit}
687               OFFSET ${limit * page}`;
688
689             const page = _.get(req, 'query.page', 0);
690             const limit = 20;
691
692             + Artist.findAll({
693               + where: {
694                 + ArtistId: 2
695               }
696             });
697
698             let sql = `SELECT * FROM artists
699               LIMIT ${limit}
700               OFFSET ${limit * page}`;
701
702             const page = _.get(req, 'query.page', 0);
703             const limit = 20;
704
705             + Artist.findAll({
706               + where: {
707                 + ArtistId: 2
708               }
709             });
710
711             let sql = `SELECT * FROM artists
712               LIMIT ${limit}
713               OFFSET ${limit * page}`;
714
715             const page = _.get(req, 'query.page', 0);
716             const limit = 20;
717
718             + Artist.findAll({
719               + where: {
720                 + ArtistId: 2
721               }
722             });
723
724             let sql = `SELECT * FROM artists
725               LIMIT ${limit}
726               OFFSET ${limit * page}`;
727
728             const page = _.get(req, 'query.page', 0);
729             const limit = 20;
730
731             + Artist.findAll({
732               + where: {
733                 + ArtistId: 2
734               }
735             });
736
737             let sql = `SELECT * FROM artists
738               LIMIT ${limit}
739               OFFSET ${limit * page}`;
740
741             const page = _.get(req, 'query.page', 0);
742             const limit = 20;
743
744             + Artist.findAll({
745               + where: {
746                 + ArtistId: 2
747               }
748             });
749
750             let sql = `SELECT * FROM artists
751               LIMIT ${limit}
752               OFFSET ${limit * page}`;
753
754             const page = _.get(req, 'query.page', 0);
755             const limit = 20;
756
757             + Artist.findAll({
758               + where: {
759                 + ArtistId: 2
760               }
761             });
762
763             let sql = `SELECT * FROM artists
764               LIMIT ${limit}
765               OFFSET ${limit * page}`;
766
767             const page = _.get(req, 'query.page', 0);
768             const limit = 20;
769
770             + Artist.findAll({
771               + where: {
772                 + ArtistId: 2
773               }
774             });
775
776             let sql = `SELECT * FROM artists
777               LIMIT ${limit}
778               OFFSET ${limit * page}`;
779
780             const page = _.get(req, 'query.page', 0);
781             const limit = 20;
782
783             + Artist.findAll({
784               + where: {
785                 + ArtistId: 2
786               }
787             });
788
789             let sql = `SELECT * FROM artists
790               LIMIT ${limit}
791               OFFSET ${limit * page}`;
792
793             const page = _.get(req, 'query.page', 0);
794             const limit = 20;
795
796             + Artist.findAll({
797               + where: {
798                 + ArtistId: 2
799               }
800             });
801
802             let sql = `SELECT * FROM artists
803               LIMIT ${limit}
804               OFFSET ${limit * page}`;
805
806             const page = _.get(req, 'query.page', 0);
807             const limit = 20;
808
809             + Artist.findAll({
810               + where: {
811                 + ArtistId: 2
812               }
813             });
814
815             let sql = `SELECT * FROM artists
816               LIMIT ${limit}
817               OFFSET ${limit * page}`;
818
819             const page = _.get(req, 'query.page', 0);
820             const limit = 20;
821
822             + Artist.findAll({
823               + where: {
824                 + ArtistId: 2
825               }
826             });
827
828             let sql = `SELECT * FROM artists
829               LIMIT ${limit}
830               OFFSET ${limit * page}`;
831
832             const page = _.get(req, 'query.page', 0);
833             const limit = 20;
834
835             + Artist.findAll({
836               + where: {
837                 + ArtistId: 2
838               }
839             });
840
841             let sql = `SELECT * FROM artists
842               LIMIT ${limit}
843               OFFSET ${limit * page}`;
844
845             const page = _.get(req, 'query.page', 0);
846             const limit = 20;
847
848             + Artist.findAll({
849               + where: {
850                 + ArtistId: 2
851               }
852             });
853
854             let sql = `SELECT * FROM artists
855               LIMIT ${limit}
856               OFFSET ${limit * page}`;
857
858             const page = _.get(req, 'query.page', 0);
859             const limit = 20;
860
861             + Artist.findAll({
862               + where: {
863                 + ArtistId: 2
864               }
865             });
866
867             let sql = `SELECT * FROM artists
868               LIMIT ${limit}
869               OFFSET ${limit * page}`;
870
871             const page = _.get(req, 'query.page', 0);
872             const limit = 20;
873
874             + Artist.findAll({
875               + where: {
876                 + ArtistId: 2
877               }
878             });
879
880             let sql = `SELECT * FROM artists
881               LIMIT ${limit}
882               OFFSET ${limit * page}`;
883
884             const page = _.get(req, 'query.page', 0);
885             const limit = 20;
886
887             + Artist.findAll({
888               + where: {
889                 + ArtistId: 2
890               }
891             });
892
893             let sql = `SELECT * FROM artists
894               LIMIT ${limit}
895               OFFSET ${limit * page}`;
896
897             const page = _.get(req, 'query.page', 0);
898             const limit = 20;
899
900             + Artist.findAll({
901               + where: {
902                 + ArtistId: 2
903               }
904             });
905
906             let sql = `SELECT * FROM artists
907               LIMIT ${limit}
908               OFFSET ${limit * page}`;
909
910             const page = _.get(req, 'query.page', 0);
911             const limit = 20;
912
913             + Artist.findAll({
914               + where: {
915                 + ArtistId: 2
916               }
917             });
918
919             let sql = `SELECT * FROM artists
920               LIMIT ${limit}
921               OFFSET ${limit * page}`;
922
923             const page = _.get(req, 'query.page', 0);
924             const limit = 20;
925
926             + Artist.findAll({
927               + where: {
928                 + ArtistId: 2
929               }
930             });
931
932             let sql = `SELECT * FROM artists
933               LIMIT ${limit}
934               OFFSET ${limit * page}`;
935
936             const page = _.get(req, 'query.page', 0);
937             const limit = 20;
938
939             + Artist.findAll({
940               + where: {
941                 + ArtistId: 2
942               }
943             });
944
945             let sql = `SELECT * FROM artists
946               LIMIT ${limit}
947               OFFSET ${limit * page}`;
948
949             const page = _.get(req, 'query.page', 0);
950             const limit = 20;
951
952             + Artist.findAll({
953               + where: {
954                 + ArtistId: 2
955               }
956             });
957
958             let sql = `SELECT * FROM artists
959               LIMIT ${limit}
960               OFFSET ${limit * page}`;
961
962             const page = _.get(req, 'query.page', 0);
963             const limit = 20;
964
965             + Artist.findAll({
966               + where: {
967                 + ArtistId: 2
968               }
969             });
970
971             let sql = `SELECT * FROM artists
972               LIMIT ${limit}
973               OFFSET ${limit * page}`;
974
975             const page = _.get(req, 'query.page', 0);
976             const limit = 20;
977
978             + Artist.findAll({
979               + where: {
980                 + ArtistId: 2
981               }
982             });
983
984             let sql = `SELECT * FROM artists
985               LIMIT ${limit}
986               OFFSET ${limit * page}`;
987
988             const page = _.get(req, 'query.page', 0);
989             const limit = 20;
990
991             + Artist.findAll({
992               + where: {
993                 + ArtistId: 2
994               }
995             });
996
997             let sql = `SELECT * FROM artists
998               LIMIT ${limit}
999               OFFSET ${limit * page}`;
1000
1001             const page = _.get(req, 'query.page', 0);
1002             const limit = 20;
1003
1004             + Artist.findAll({
1005               + where: {
1006                 + ArtistId: 2
1007               }
1008             });
1009
1010             let sql = `SELECT * FROM artists
1011               LIMIT ${limit}
1012               OFFSET ${limit * page}`;
1013
1014             const page = _.get(req, 'query.page', 0);
1015             const limit = 20;
1016
1017             + Artist.findAll({
1018               + where: {
1019                 + ArtistId: 2
1020               }
1021             });
1022
1023             let sql = `SELECT * FROM artists
1024               LIMIT ${limit}
1025               OFFSET ${limit * page}`;
1026
1027             const page = _.get(req, 'query.page', 0);
1028             const limit = 20;
1029
1030             + Artist.findAll({
1031               + where: {
1032                 + ArtistId: 2
1033               }
1034             });
1035
1036             let sql = `SELECT * FROM artists
1037               LIMIT ${limit}
1038               OFFSET ${limit * page}`;
1039
1040             const page = _.get(req, 'query.page', 0);
1041             const limit = 20;
1042
1043             + Artist.findAll({
1044               + where: {
1045                 + ArtistId: 2
1046               }
1047             });
1048
1049             let sql = `SELECT * FROM artists
1050               LIMIT ${limit}
1051               OFFSET ${limit * page}`;
1052
1053             const page = _.get(req, 'query.page', 0);
1054             const limit = 20;
1055
1056             + Artist.findAll({
1057               + where: {
1058                 + ArtistId: 2
1059               }
1060             });
1061
1062             let sql = `SELECT * FROM artists
1063               LIMIT ${limit}
1064               OFFSET ${limit * page}`;
1065
1066             const page = _.get(req, 'query.page', 0);
1067             const limit = 20;
1068
1069             + Artist.findAll({
1070               + where: {
1071                 + ArtistId: 2
1072               }
1073             });
1074
1075             let sql = `SELECT * FROM artists
1076               LIMIT ${limit}
1077               OFFSET ${limit * page}`;
1078
1079             const page = _.get(req, 'query.page', 0);
1080             const limit = 20;
1081
1082             + Artist.findAll({
1083               + where: {
1084                 + ArtistId: 2
1085               }
1086             });
1087
1088             let sql = `SELECT * FROM artists
1089               LIMIT ${limit}
1090               OFFSET ${limit * page}`;
1091
1092             const page = _.get(req, 'query.page', 0);
1093             const limit = 20;
1094
1095             + Artist.findAll({
1096               + where: {
1097                 + ArtistId: 2
1098               }
1099             });
1100
1101             let sql = `SELECT * FROM artists
1102               LIMIT ${limit}
1103               OFFSET ${limit * page}`;
1104
1105             const page = _.get(req, 'query.page', 0);
1106             const limit = 20;
1107
1108             + Artist.findAll({
1109               + where: {
1110                 + ArtistId: 2
1111               }
1112             });
1113
1114             let sql = `SELECT * FROM artists
1115               LIMIT ${limit}
1116               OFFSET ${limit * page}`;
1117
1118             const page = _.get(req, 'query.page', 0);
1119             const limit = 20;
1120
1121             + Artist.findAll({
1122               + where: {
1123                 + ArtistId: 2
1124               }
1125             });
1126
1127             let sql = `SELECT * FROM artists
1128               LIMIT ${limit}
1129               OFFSET ${limit * page}`;
1130
1131             const page = _.get(req, 'query.page', 0);
1132             const limit = 20;
1133
1134             + Artist.findAll({
1135               + where: {
1136                 + ArtistId: 2
1137               }
1138             });
1139
1140             let sql = `SELECT * FROM artists
1141               LIMIT ${limit}
1142               OFFSET ${limit * page}`;
1143
1144             const page = _.get(req, 'query.page', 0);
1145             const limit = 20;
1146
1147             + Artist.findAll({
1148               + where: {
1149                 + ArtistId: 2
1150               }
1151             });
1152
1153             let sql = `SELECT * FROM artists
1154               LIMIT ${limit}
1155               OFFSET ${limit * page}`;
1156
1157             const page = _.get(req, 'query.page', 0);
1158             const limit = 20;
1159
1160             + Artist.findAll({
1161               + where: {
1162                 + ArtistId: 2
1163               }
1164             });
1165
1166             let sql = `SELECT * FROM artists
1167               LIMIT ${limit}
1168               OFFSET ${limit * page}`;
1169
1170             const page = _.get(req, 'query.page', 0);
1171             const limit = 20;
1172
1173             + Artist.findAll({
1174               + where: {
1175                 + ArtistId: 2
1176               }
1177             });
1178
1179             let sql = `SELECT * FROM artists
1180               LIMIT ${limit}
1181               OFFSET ${limit * page}`;
1182
1183             const page = _.get(req, 'query.page', 0);
1184             const limit = 20;
1185
1186             + Artist.findAll({
1187               + where: {
1188                 + ArtistId: 2
1189               }
1190             });
1191
1192             let sql = `SELECT * FROM artists
1193               LIMIT ${limit}
1194               OFFSET ${limit * page}`;
1195
1196             const page = _.get(req, 'query.page', 0);
1197             const limit = 20;
1198
1199             + Artist.findAll({
1200               + where: {
1201                 + ArtistId: 2
1202               }
1203             });
1204
1205             let sql = `SELECT * FROM artists
1206               LIMIT ${limit}
1207               OFFSET ${limit * page}`;
1208
1209             const page = _.get(req, 'query.page', 0);
1210             const limit = 20;
1211
1212             + Artist.findAll({
1213               + where: {
1214                 + ArtistId: 2
1215               }
1216             });
1217
1218             let sql = `SELECT * FROM artists
1219               LIMIT ${limit}
1220               OFFSET ${limit * page}`;
1221
1222             const page = _.get(req, 'query.page', 0);
1223             const limit = 20;
1224
1225             + Artist.findAll({
1226               + where: {
1227                 + ArtistId: 2
1228               }
1229             });
1230
1231             let sql = `SELECT * FROM artists
1232               LIMIT ${limit}
1233               OFFSET ${limit * page}`;
1234
1235             const page = _.get(req, 'query.page', 0);
1236             const limit = 20;
1237
1238             + Artist.findAll({
1239               + where: {
1240                 + ArtistId: 2
1241               }
1242             });
1243
1244             let sql = `SELECT * FROM artists
1245               LIMIT ${limit}
1246               OFFSET ${limit * page}`;
1247
1248             const page = _.get(req, 'query.page', 0);
1249             const limit = 20;
1250
1251             + Artist.findAll({
1252               + where: {
1253                 + ArtistId: 2
1254               }
1255             });
1256
1257             let sql = `SELECT * FROM artists
1258               LIMIT ${limit}
1259               OFFSET ${limit * page}`;
1260
1261             const page = _.get(req, 'query.page', 0);
1262             const limit = 20;
1263
1264             + Artist.findAll({
1265               + where: {
1266                 + ArtistId: 2
1267               }
1268             });
1269
1270             let sql = `SELECT * FROM artists
1271               LIMIT ${limit}
1272               OFFSET ${limit * page}`;
1273
1274             const page = _.get(req, 'query.page', 0);
1275             const limit = 20;
1276
1277             + Artist.findAll({
1278               + where: {
1279                 + ArtistId: 2
1280               }
1281             });
1282
1283             let sql = `SELECT * FROM artists
1284               LIMIT ${limit}
1285               OFFSET ${limit * page}`;
1286
1287             const page = _.get(req, 'query.page', 0);
1288             const limit = 20;
1289
1290             + Artist.findAll({
1291               + where: {
1292                 + ArtistId: 2
1293               }
1294             });
1295
1296             let sql = `SELECT * FROM artists
1297               LIMIT ${limit}
1298               OFFSET ${limit * page}`;
1299
1300             const page = _.get(req, 'query.page', 0);
1301             const limit = 20;
1302
1303             + Artist.findAll({
1304               + where: {
1305                 + ArtistId: 2
1306               }
1307             });
1308
1309             let sql = `SELECT * FROM artists
1310               LIMIT ${limit}
1311               OFFSET ${limit * page}`;
1312
1313             const page = _.get(req, 'query.page', 0);
1314             const limit = 20;
1315
1316             + Artist.findAll({
1317               +
```

Default branch		
master	Updated 1 minute ago by tinchooviedo	Default
Your branches		
ramaGit	Updated 2 hours ago by tinchooviedo	1 0 #1 Merged
Active branches		
ramaGit	Updated 2 hours ago by tinchooviedo	1 0 #1 Merged

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 repositorio 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, así 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.

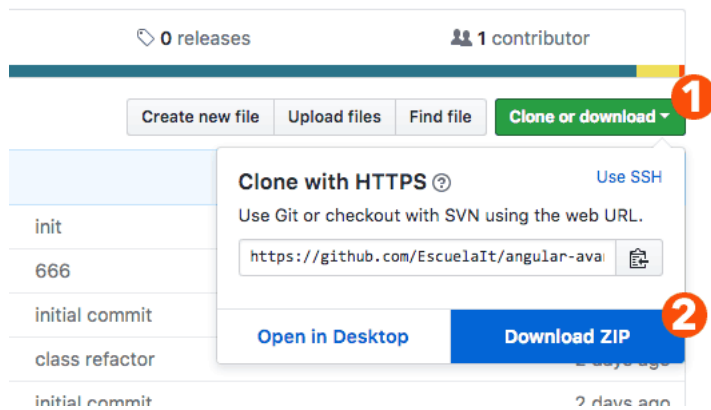
CLONAR UN REPOSITORIO

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:

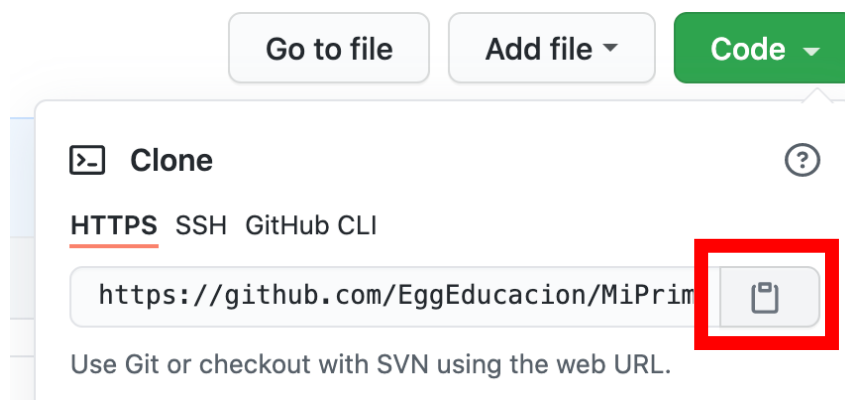
- No crea un repositorio Git en local con los cambios que el repositorio remoto ha tenido a lo largo del tiempo. Es decir, te descargas el código, pero nada más.
- No podrás luego enviar cambios al repositorio remoto, una vez los hayas realizado en local.

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)
```

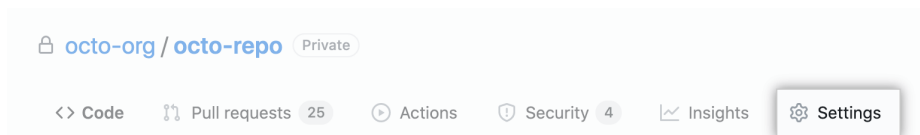
De esta manera nosotros ya tenemos el repositorio remoto para trabajar local y podremos hacer los cambios que queramos y subir los cambios con los comandos que explicamos previamente.

INVITAR COLABORADORES A UN REPOSITORIO PERSONAL

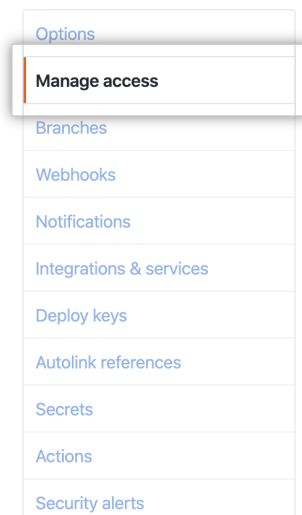
Nosotros vimos como clonar un repositorio para poder usar su código y si quisiéramos hacer cambios y subir esos cambios. Todos los usuarios de GitHub pueden ver y clonar tu repositorio, siempre y cuando sea un repositorio publico. Pero, no todo las personas que clonan tu repositorio pueden subir sus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

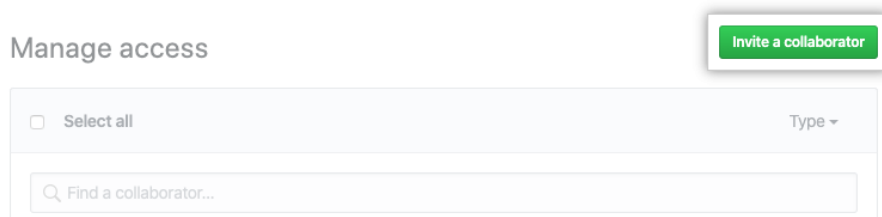
1. Solicita el nombre de usuario de la persona que estás invitando como colaboradora.
2. En GitHub, visita la página principal del repositorio.
3. Debajo de tu nombre de repositorio, da clic en **Configuración**.



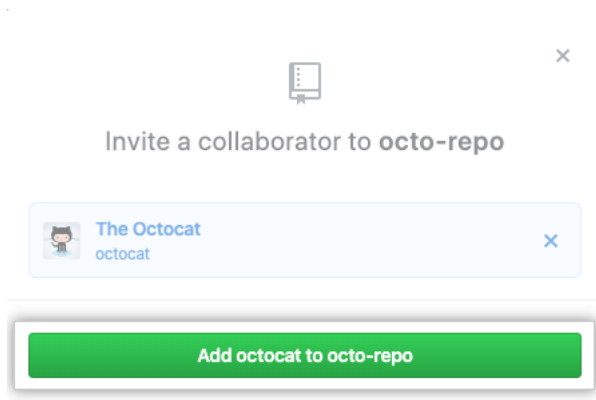
4. En la barra lateral izquierda, da clic en **Administrar acceso**.



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.

PREGUNTAS DE APRENDIZAJE

1) ¿Qué es Git?

- a) Una plataforma de repositorios remotos
- b) Un nombre para GitHub
- c) Un lenguaje de programación
- d) Un sistema de control de versiones

2) ¿Qué es GitHub?

- a) Una plataforma de repositorios remotos
- b) Un nombre para GitHub
- c) Un lenguaje de programación
- d) Un sistema de control de versiones

3) ¿Que comando usamos para iniciar un repositorio de manera local?

- a) Git clone
- b) Git commit
- c) Git init
- d) Git status

4) ¿Que comando usamos para obtener la información de nuestra rama?

- a) Git help
- b) Git add
- c) Git info
- d) Git status

5) ¿Que comando usamos para incluir los cambios en el repositorio local?

- a) Git add
- b) Git clone
- c) Git status
- d) Git init

6) ¿Que comando usamos para guardar nuestros cambios en el repositorio local?

- a) Git clone
- b) Git save
- c) Git commit
- d) Git add

7) ¿Que comando usamos para enviar nuestros cambios al repositorio remoto?

- a) Git send
- b) Git push
- c) Git pull
- d) Git remote

8) ¿Que comando usamos para crear una rama?

- a) Git branch
- b) Git checkout -b
- c) Git clone
- d) Git init

9) ¿Que comando usamos para unir los cambios de dos ramas?

- a) Git merge
- b) Git join
- c) Git pull
- d) Git push

10) Para unir nuestros cambios de dos ramas en el repositorio remoto vamos a usar:

- a) Pull Request
- b) Git Merge remote
- c) Pull Merge
- d) Ninguna de las anteriores

11) ¿Que comando usamos para clonar un repositorio de GitHub?

- a) Git pull
- b) Git clone
- c) Git download
- d) Git remote

EJERCICIOS DE APRENDIZAJE

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

VER VIDEOS:

- A. [Fundamentos Básicos de Git](#)
- B. [Inicialización Repositorio Local](#)
- C. [Enviar Información entre Repositorios](#)
- D. [Moverse entre Diferentes Commits](#)

1. Vamos a crear una carpeta con un archivo txt, dentro poner el texto que queramos, esta carpeta junto con el txt tienen que iniciarse como un repositorio local. Además deberemos subir este archivo a un repositorio remoto.
2. Ahora tenemos que modificar el txt y subir esos cambios al repositorio remoto.

VER VIDEOS:

- A. [Creación de una Nueva Rama](#)
- B. [Merge entre Ramas](#)
- C. [Clonar un Repositorio](#)
- D. [Repaso de Comandos](#)

3. 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 dirige 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 nombres.
4. 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 vacío 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.