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 descargarte la aplicación, sino también entrar a su perfil para leer sobre ella o colaborar con su desarrollo.

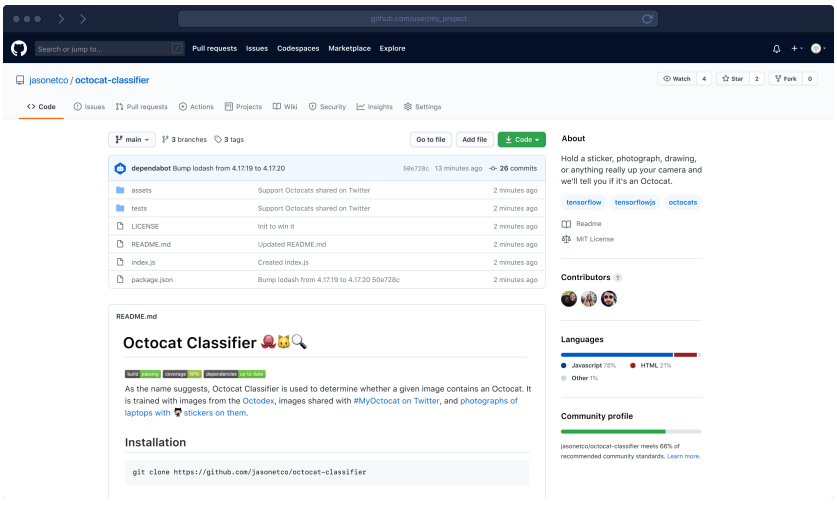
Un sistema de gestión de versiones es ese con el que **los desarrolladores pueden administrar su proyecto**, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán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

Git, al ser un sistema de control, va 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?

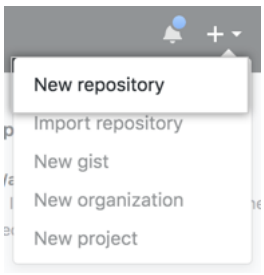
Github permite que los desarrolladores alojen proyectos creando repositorios de forma gratuita. Un repositorio es como una carpeta para tu proyecto. El repositorio de tu proyecto contiene todos los archivos de tu repositorio y almacena el historial de revisión de cada archivo. También puedes debatir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



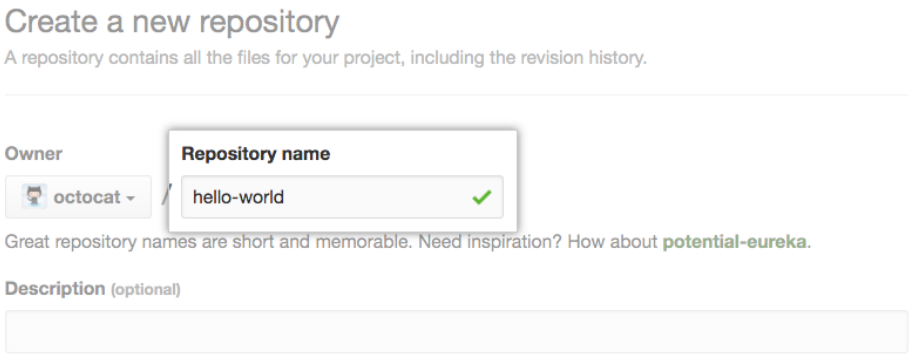
CREAR UN REPOSITORIO EN GITHUB

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

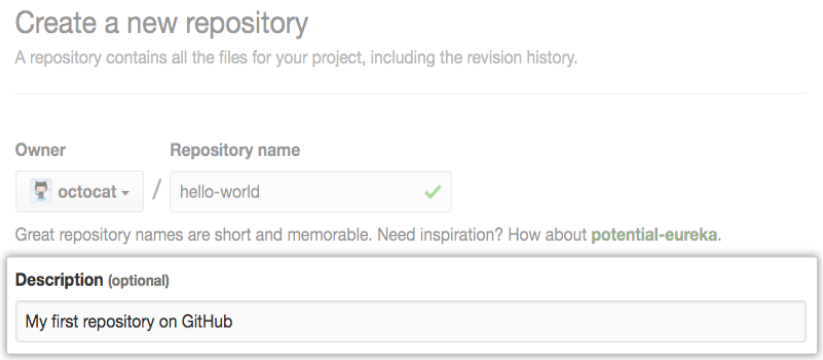
1) En la esquina superior derecha de cualquier página, utiliza el menú desplegable + y selecciona Repositorio Nuevo



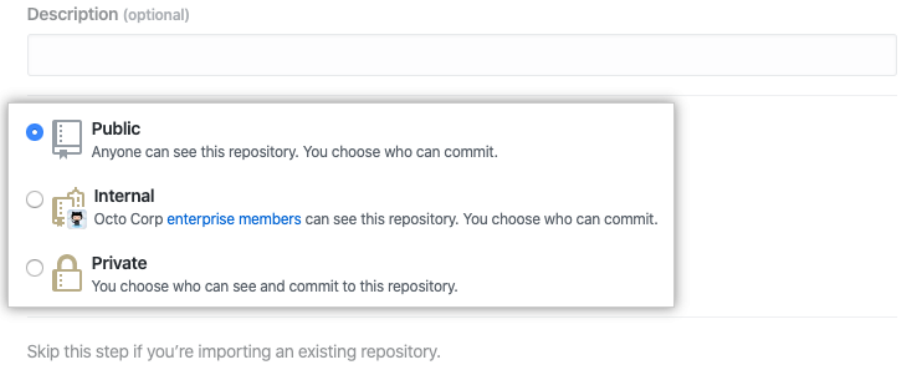
2) Escribe un nombre corto y fácil de recordar para tu repositorio. Por ejemplo: "hola-mundo".



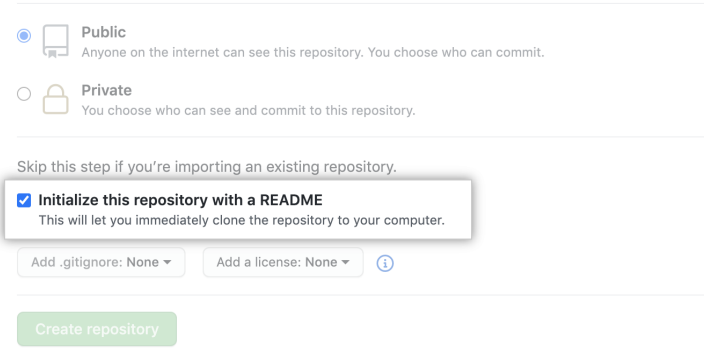
3) También puedes agregar una descripción de tu repositorio. Por ejemplo, "Mi primer repositorio en GitHub".



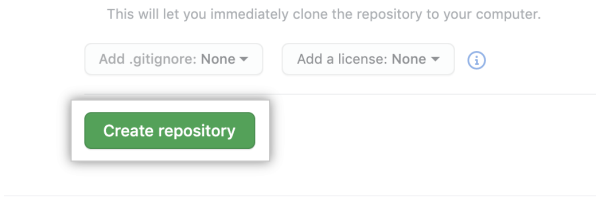
4) Elige la visibilidad del repositorio. Puedes restringir quién tiene acceso a un repositorio eligiendo la visibilidad de un repositorio: público o privado. Publico significa que cualquier persona puede ver ese repositorio y privado significa que solo personas autorizadas pueden verlo. Que sea publico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



5) Podemos crear el repositorio con un ReadMe



6) Da click en Crear repositorio.



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.

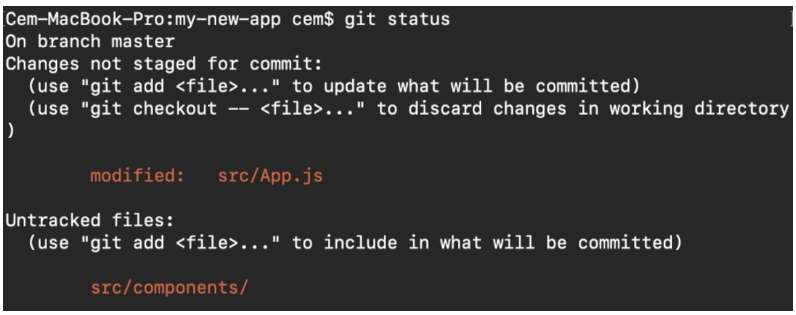
Git Status

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



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.

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

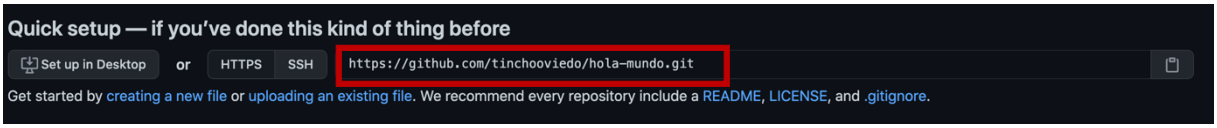
Primero deberemos crear un repositorio, sin el archivo readME.

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 las 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.

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.

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:

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"
```

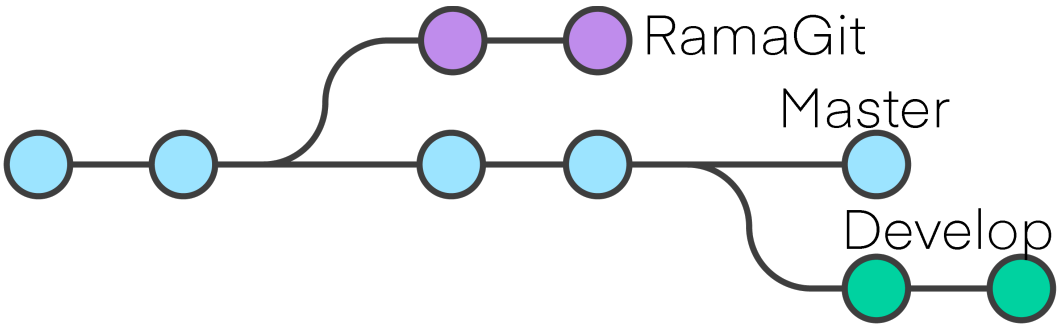
No vamos a hacer un push todavía porque eso lo vamos a explicar más adelante, ya que eso hará que nuestra rama aparezca en el repositorio remoto.

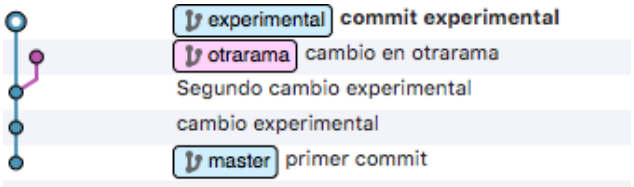
Habiendo hecho un commit en nuestra nueva rama, observarás que al hacer el show-branches te mostrará nuevos datos:

```
[→  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.

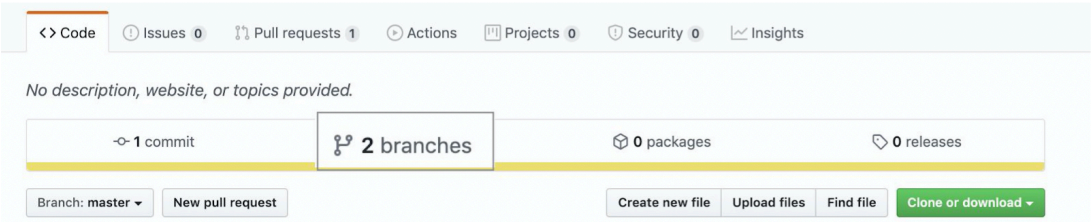
Esto sería asi:

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

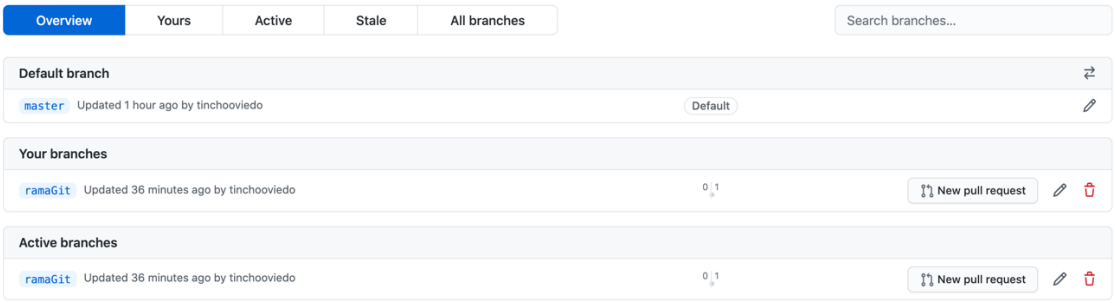
Una vez esto hecho esto podríamos pararnos en nuestra rama y simplemente escribir:

```
git push
```

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



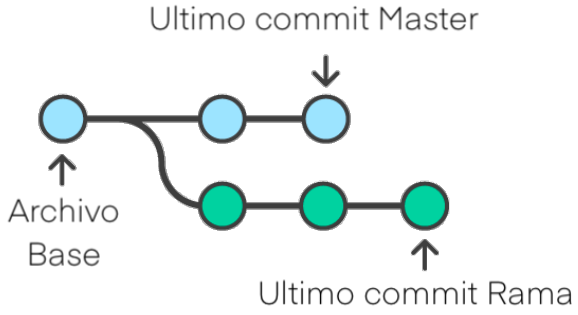
Y ahí podremos ver las dos ramas

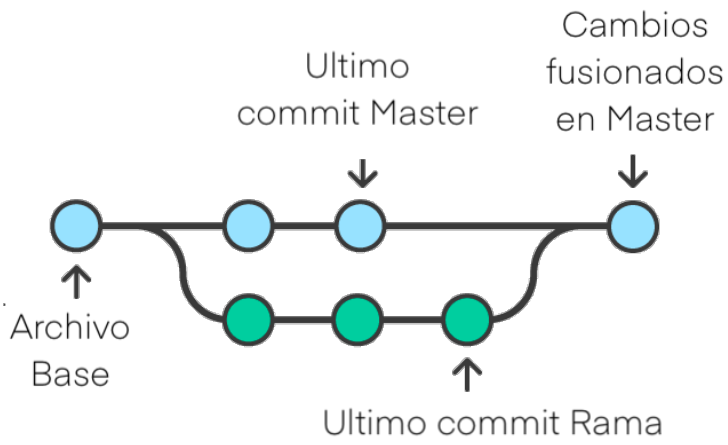


Puedes subir tanto commits creas convenientes a tu branch antes de mergear a master, siempre es mejor pequeños y frecuentes cambios que pocos y grandes.

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"
```

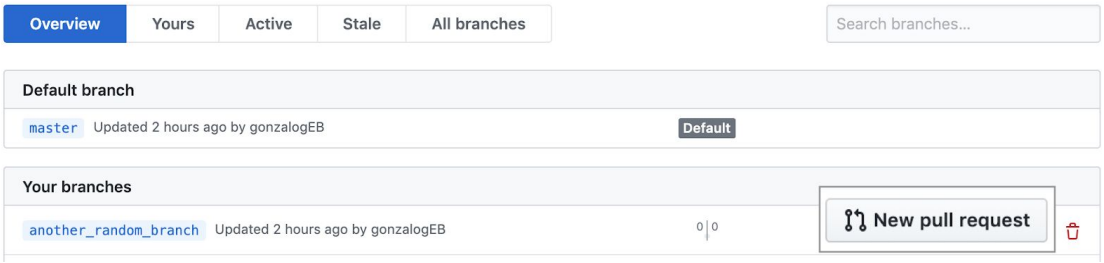
Luego podremos comprobar que nuestra rama master tiene todo el código nuevo de la ramaGit y podremos hacer nuevos commits en master para seguir el desarrollo de nuestro proyecto ya con la rama principal, si es nuestro deseo.

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**

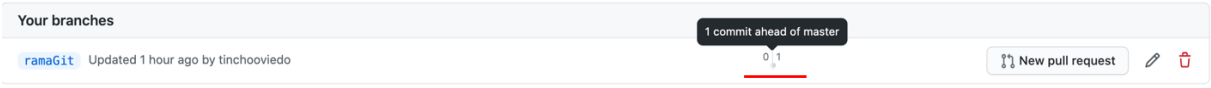
Vamos a pararnos de nuevo en una etapa en donde no hemos mergeado las ramas. Hasta ese punto habíamos logrado independizar nuestros cambios de los del resto del equipo, pero se acercaba la hora de publicar nuestros cambios y surge la necesidad de conocer y/o validar cuan diferente es nuestra versión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

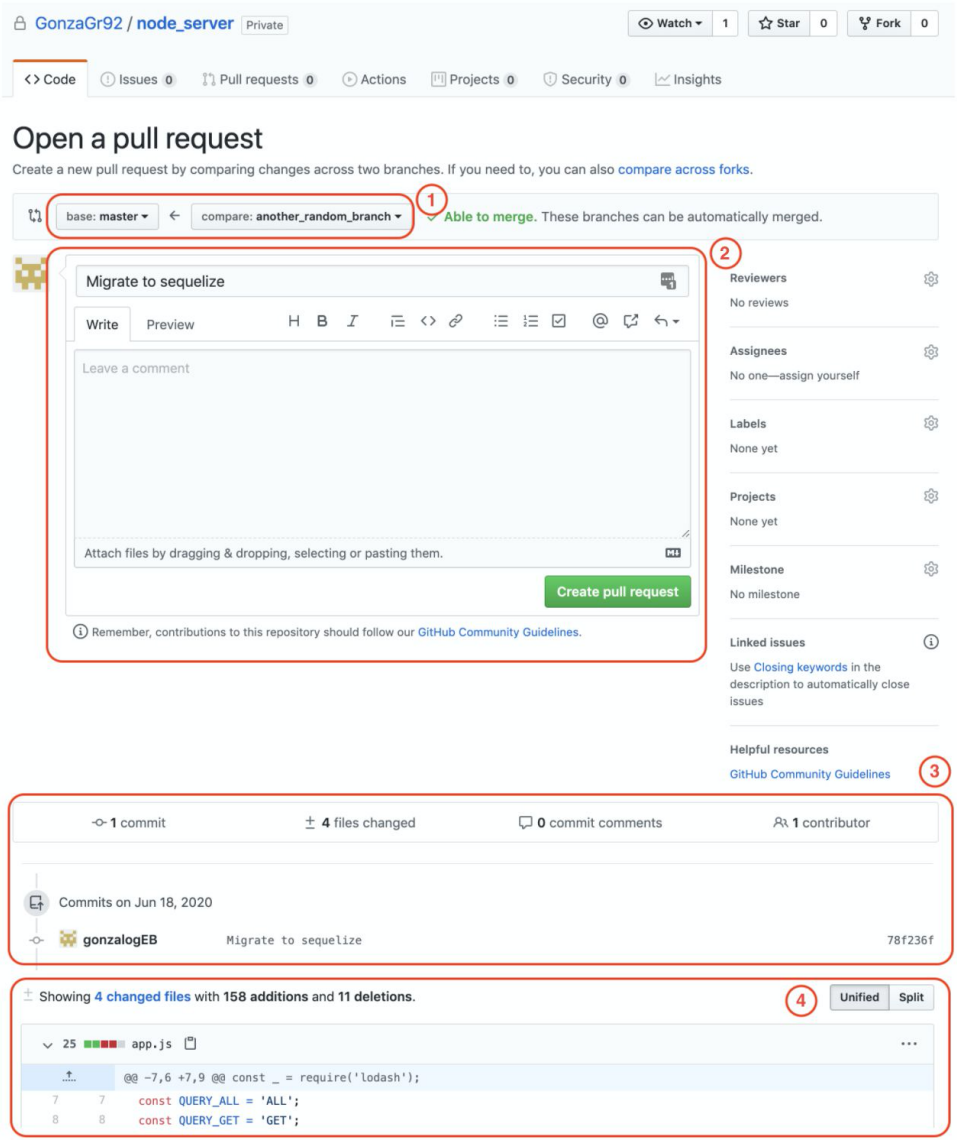


Antes de hacer nuestro pull request, podemos ver, cuantos commits (cambios) son los que separan a otra rama de master. Si nos fijamos nos salen dos ceros, pero si master estuviera un commit por delante de alguna rama saldrían un uno en el cero de la izquierda y asi se incrementa el numero según la cantidad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

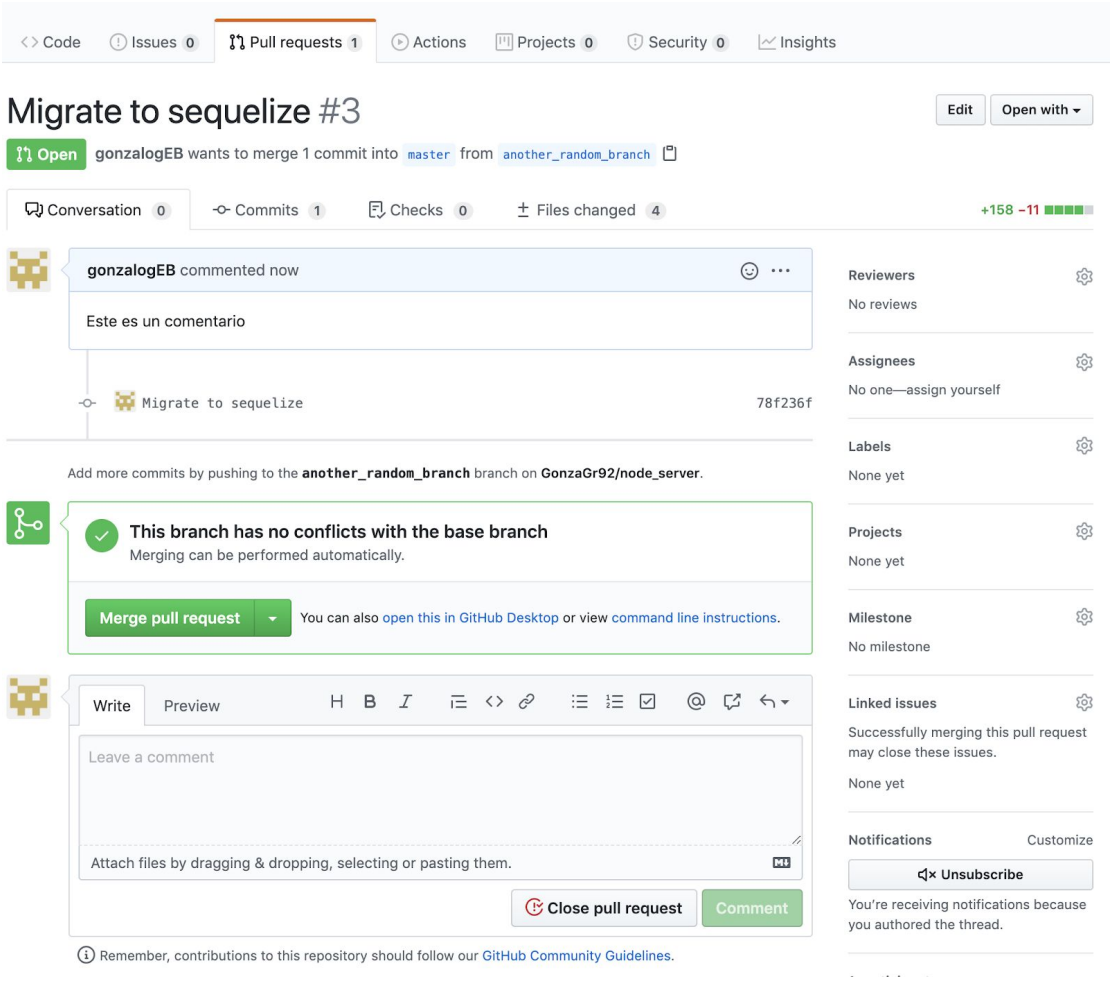


Nos mostrará algo similar a lo siguiente.

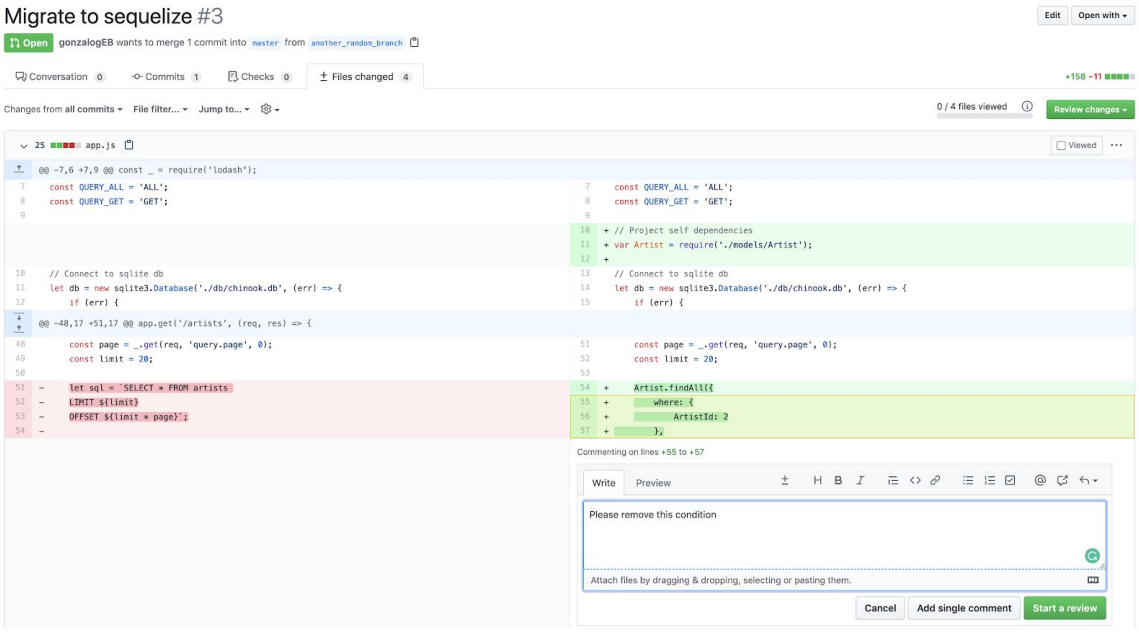
Referencias:

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.

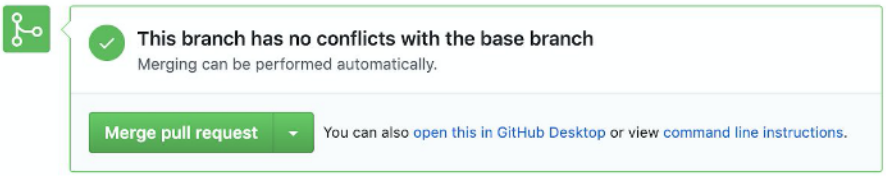


Por otro lado podemos usar la pestaña de “Files changed” para hacer code review.

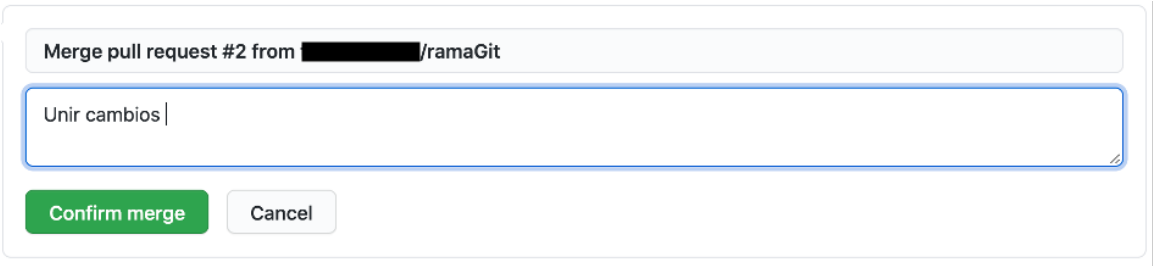


Si detectamos alguna línea de código que requiera cambios puedes clickear sobre ella y agregar un comentario para que el autor del pull request lo modifique. No es necesario volver a crear un nuevo pull request para actualizar los cambios, simplemente haciendo un commit sobre el branch es suficiente,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

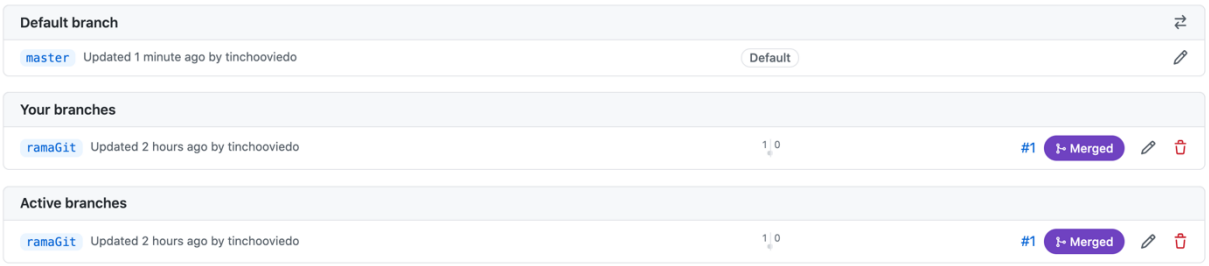
Si esta todo bien y no hay conflictos podemos mergear nuestro branch a master clickeando en el botón “Merge pull request” y de eso modo finaliza el ciclo del branch.



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.

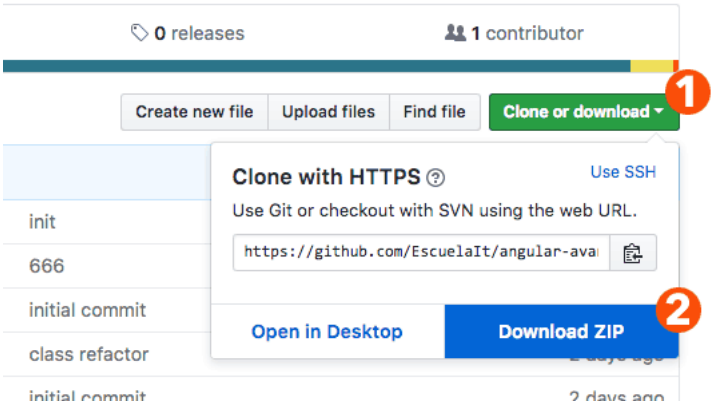
CLONAR UN REPOSITORIO

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:

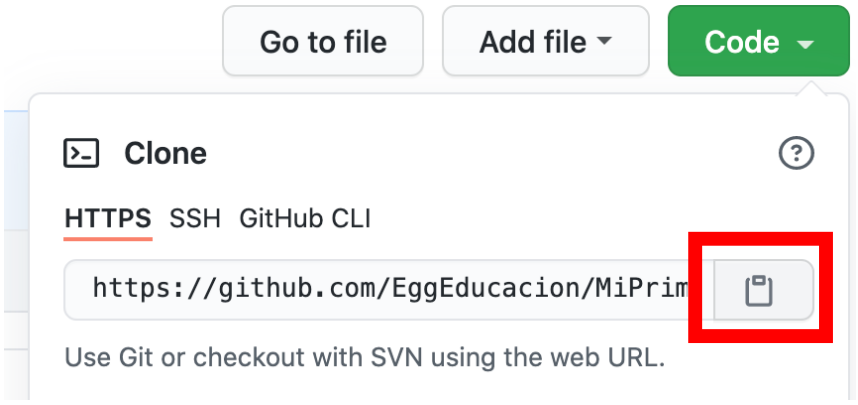
- No crea un repositorio Git en local con los cambios que el repositorio remoto ha tenido a lo largo del tiempo. Es decir, te descargas el código, pero nada más.
- No podrás luego enviar cambios al repositorio remoto, una vez los hayas realizado en local.

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)
```

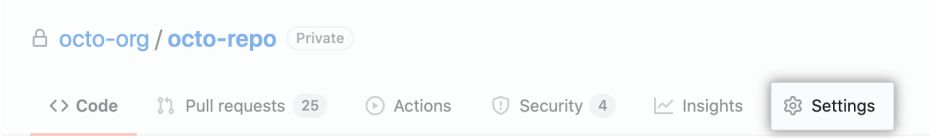
De esta manera nosotros ya tenemos el repositorio remoto para trabajar local y podremos hacer los cambios que queramos y subir los cambios con los comandos que explicamos previamente.

INVITAR COLABORADORES A UN REPOSITORIO PERSONAL

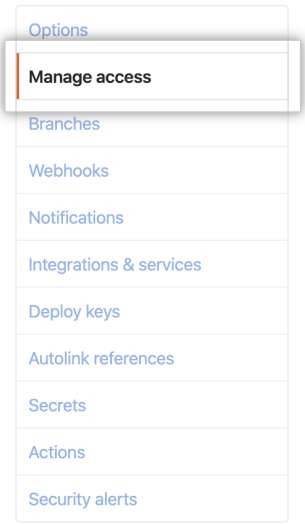
Nosotros vimos como clonar un repositorio para poder usar su código y si quisiéramos hacer cambios y subir esos cambios. Todos los usuarios de GitHub pueden ver y clonar tu repositorio, siempre y cuando sea un repositorio publico. Pero, no todo las personas que clonan tu repositorio pueden subir sus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

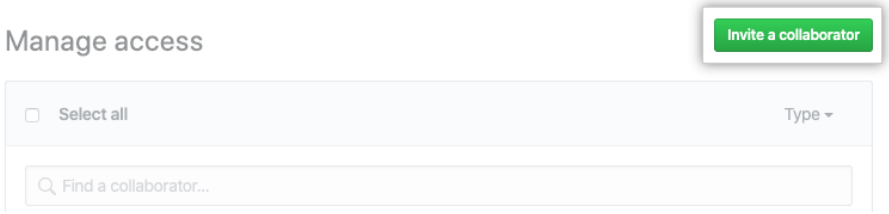
1. Solicita el nombre de usuario de la persona que estás invitando como colaboradora.
2. En GitHub, visita la página principal del repositorio.
3. Debajo de tu nombre de repositorio, da clic en Configuración.



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

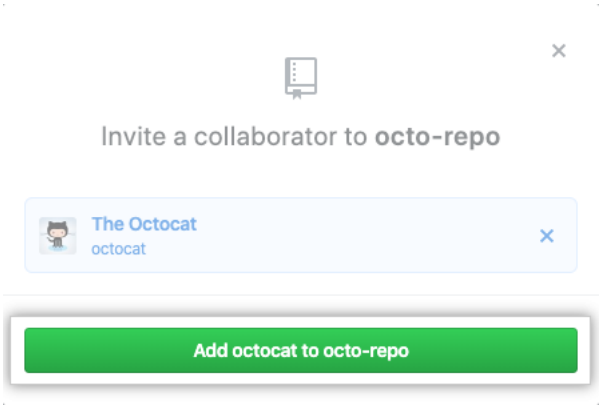


5. Da clic en Invitar un colaborador.



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.

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 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.

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 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.