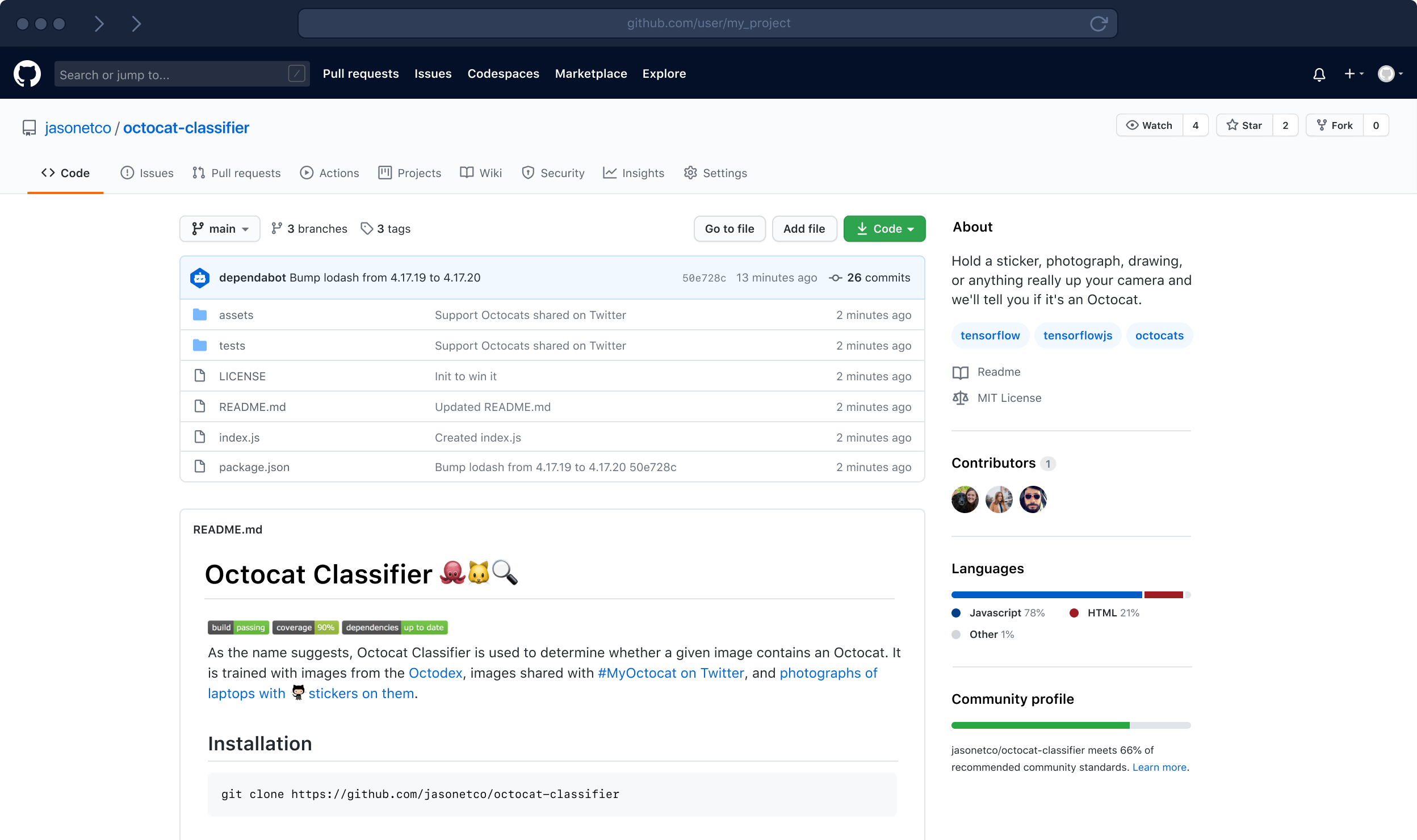
**Descargar Git e Instalar**

<https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git>

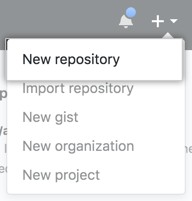
Github permite que los desarrolladores alojen proyectos creando repositorios de forma gratuita. Un repositorio es como una carpeta para tu proyecto. El repositorio de tu proyecto contiene todos los archivos de tu repositorio y almacena el historial de revisión de cada archivo. También puedes debatir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



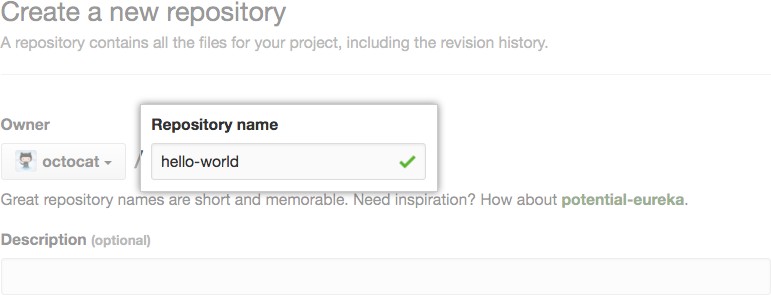
**CREAR UN REPOSITORIO EN GITHUB**

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

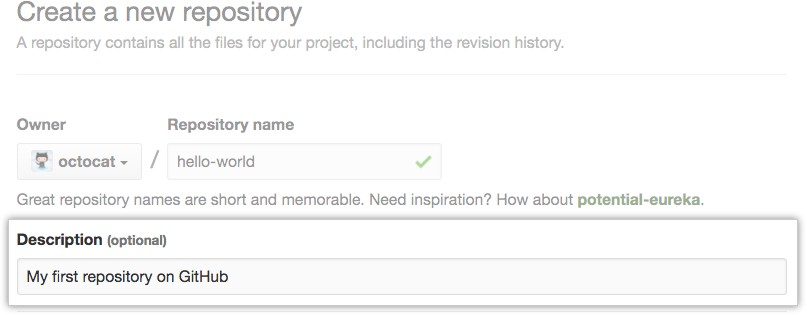
En la esquina superior derecha de cualquier página, utiliza el menú desplegable + y selecciona Repositorio Nuevo

. 

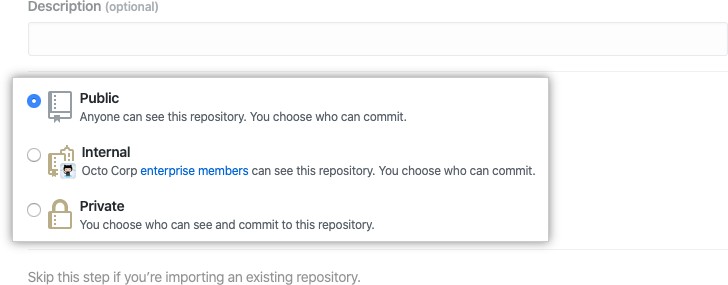
Escribe un nombre corto y fácil de recordar para tu repositorio. Por ejemplo: "hola- mundo".



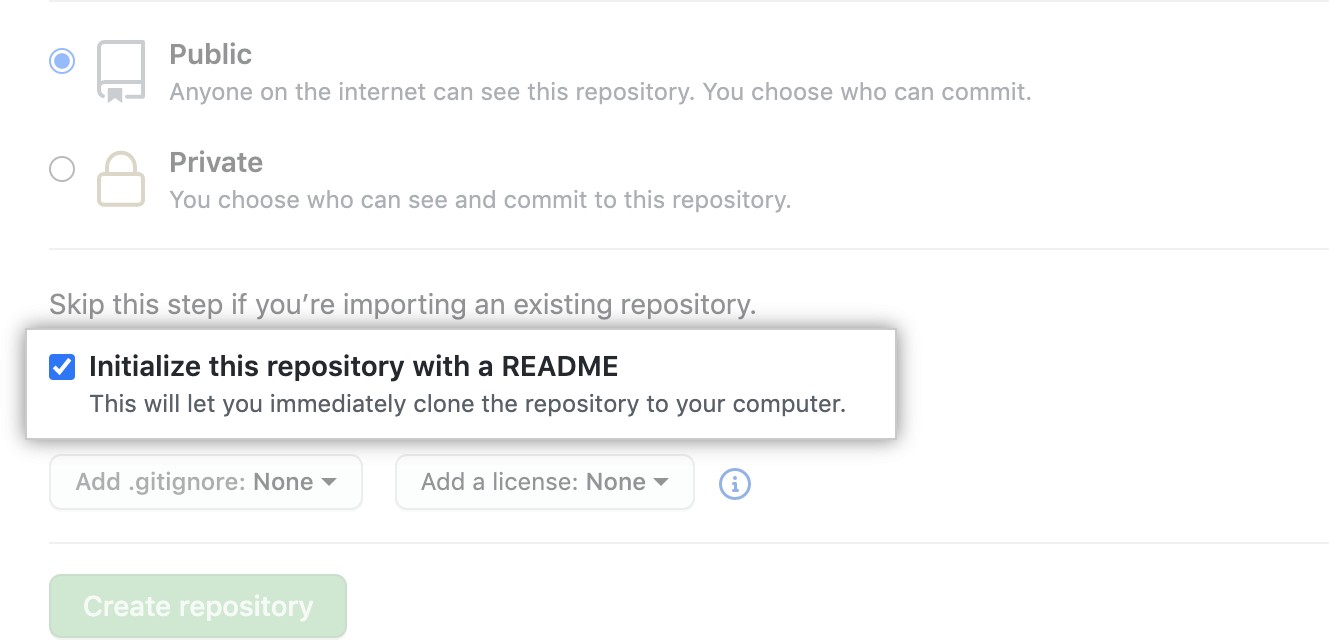
También puedes agregar una descripción de tu repositorio. Por ejemplo, "Mi primer repositorio en GitHub".



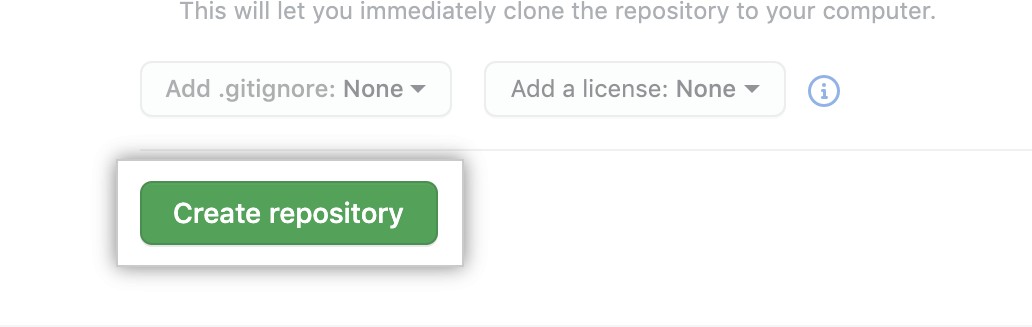
Elige la visibilidad del repositorio. Puedes restringir quién tiene acceso a un repositorio eligiendo la visibilidad de un repositorio: público o privado. Publico significa que cualquier persona puede ver ese repositorio y privado significa que solo personas autorizadas pueden verlo. Que sea publico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



Podemos crear el repositorio con un ReadMe



Da click en Crear repositorio.



**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"](mailto: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](mailto:user.email%3Dmicorreopersonal@juan.com)

**FORMAS DE COMENZAR A TRABAJAR CON GIT**

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

Trabajar en local, en un repositorio que me cree en mi máquina y vincularlo a un repositorio creado en GitHub

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

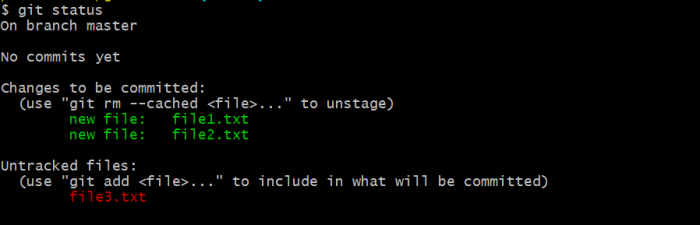
Podemos encontrar información como:

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



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:

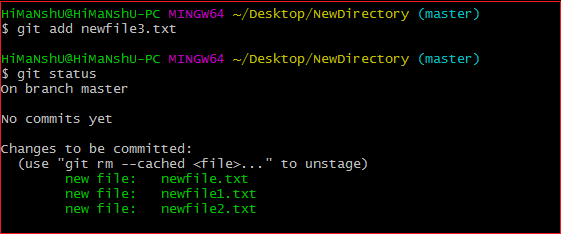
$ git add <archivo>

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**

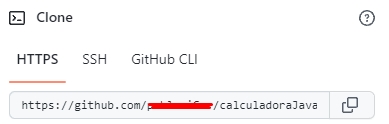
Primero deberemos crear un repositorio, sin el archivo readME.

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:



**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".

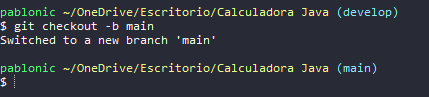
Recordemos que en GitHub esta rama puede llamarse Main, siempre podemos cambiar el nombre de la rama a Master con las configuraciones de GitHub.

**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.

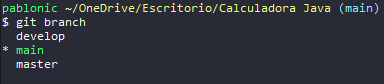
##### git checkout -b nombre\_de\_tu\_branch

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



Para ver que ramas tenemos pondremos el comando:

git branch

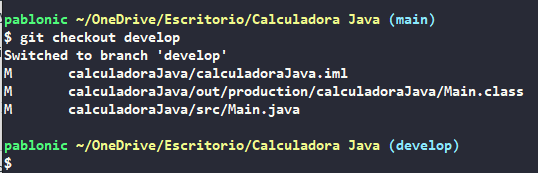


**PASAR DE UNA RAMA A OTRA**

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:

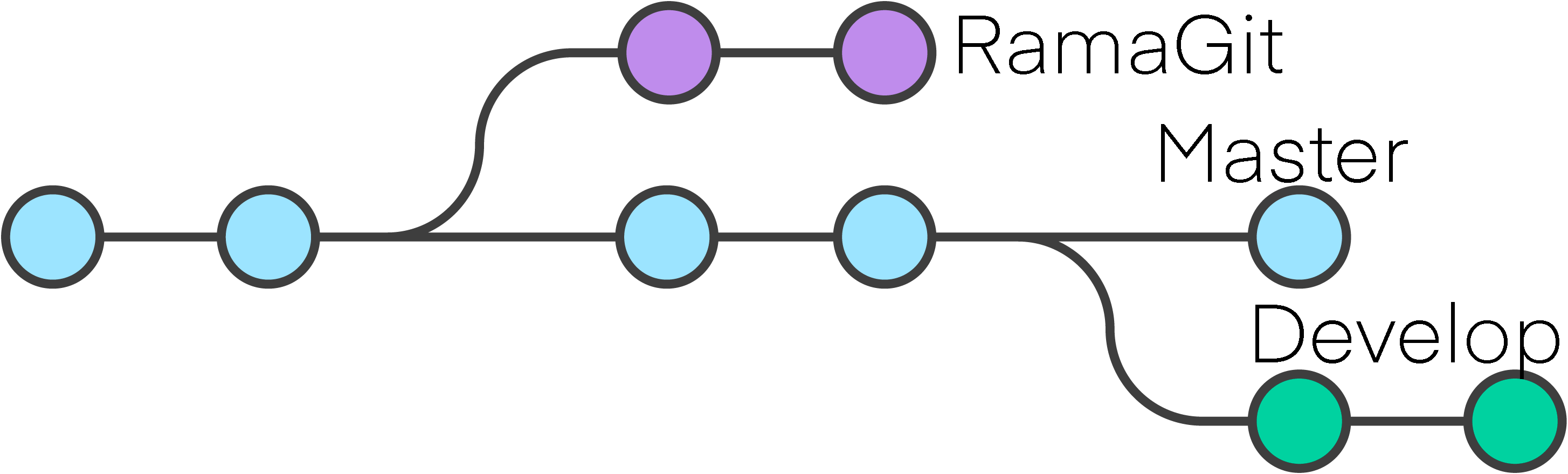
##### git add nombre\_del\_archivo (punto(.) en lugar del nombre si quieres agregar todos cambiados)

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.

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.

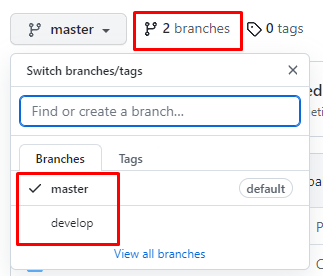
Esto sería asi:

##### git push -u origin nombre\_de\_tu\_branch

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

##### git push

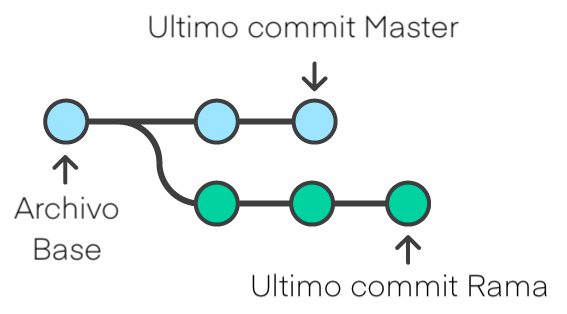
Una vez que hagamos para ver las ramas, primero iremos a branches:

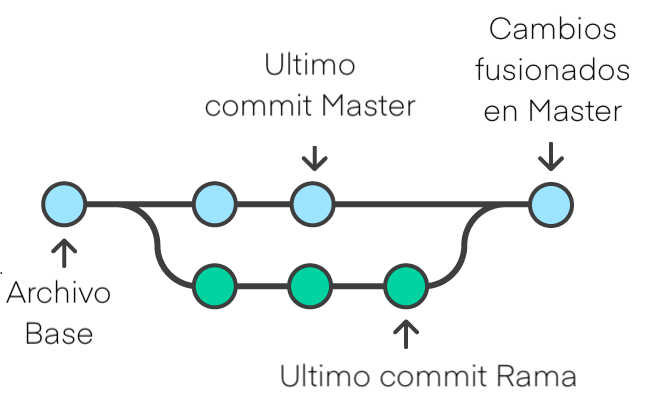


Puedes subir tanto commits creas convenientes a tu branch antes de mergear a master, siempre es mejor pequeños y frecuentes cambios que pocos y grandes.

**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”

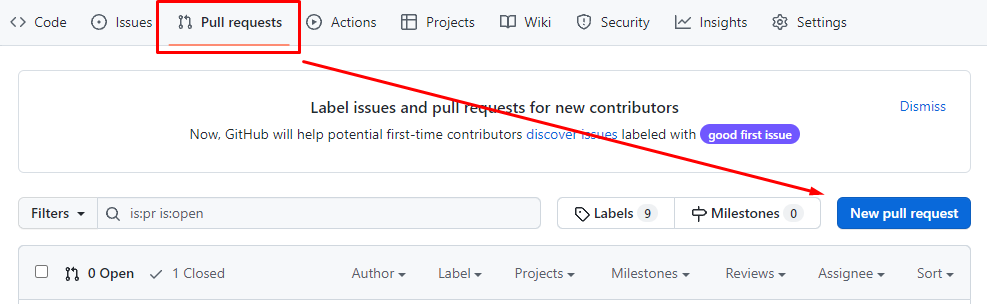
Luego podremos comprobar que nuestra rama master tiene todo el código nuevo de la ramaGit y podremos hacer nuevos commits en master para seguir el desarrollo de nuestro proyecto ya con la rama principal, si es nuestro deseo.

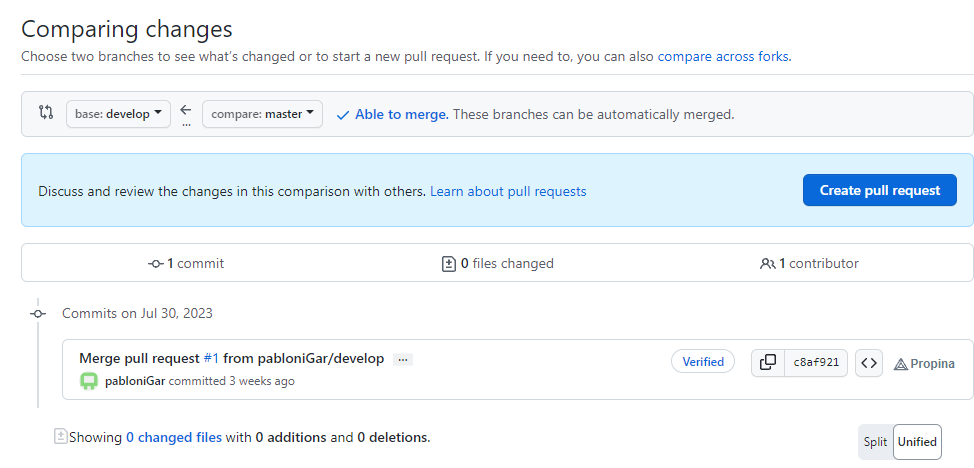
**PULL REQUEST**

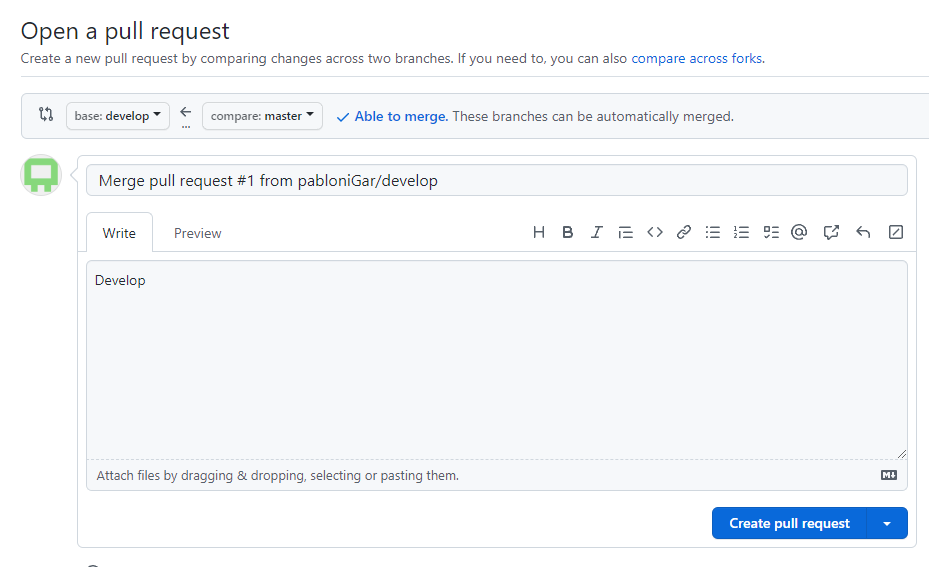
Previamente habíamos fusionado nuestras ramas a través del comando merge, pero GitHub nos permite fusionar nuestras ramas y además ver los cambios que hay entre una rama y otra, gracias al Pull Request

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.





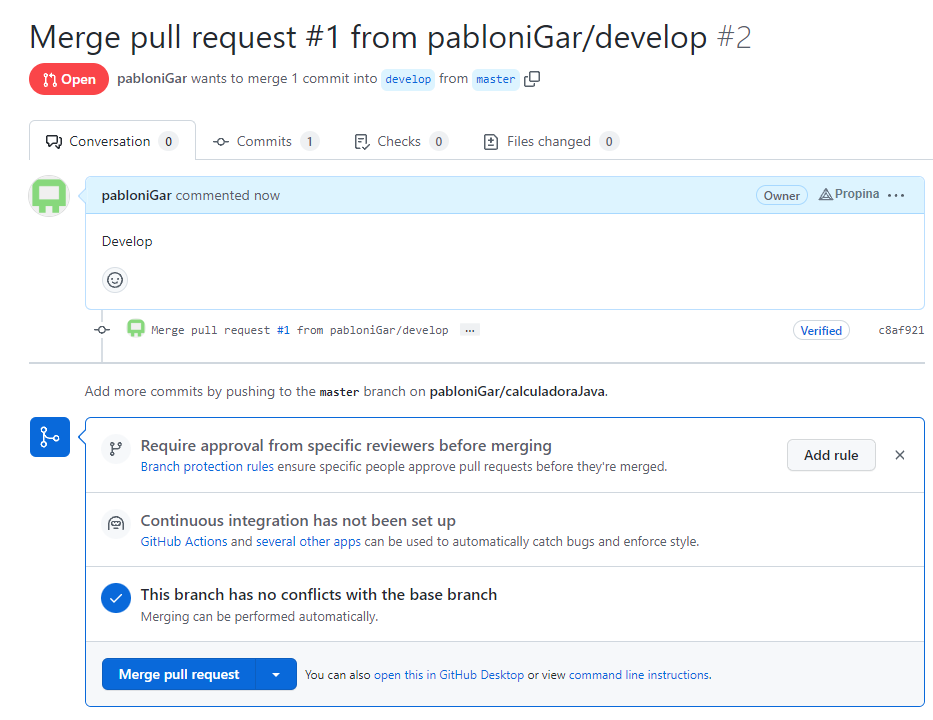


Nos mostrará algo similar a lo siguiente.

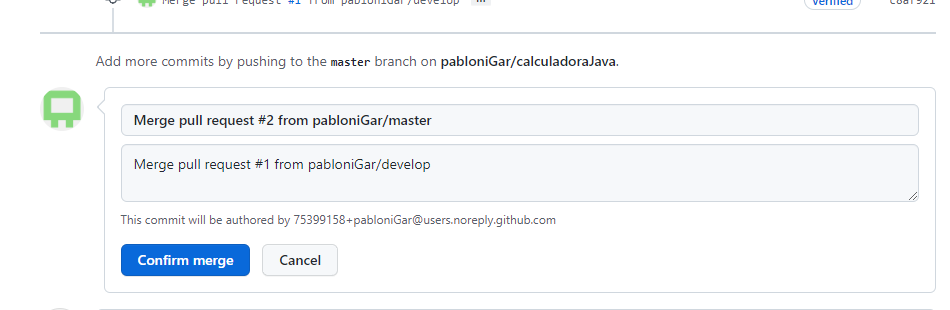
Referencias:

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.



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.



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



**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.