Comandos Git:

**CONFIGURACIÓN INICIAL**

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

git config -—global user.name “Jhoel Perez”

Ahora indique el correo electrónico del usuario para git:

git config --global user. email [micorreopersonal@jhoel.com](mailto:micorreopersonal@jhoel.com)

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

git init: Es para crear nuestro repositorio local (la carpeta .git, o sea la master branch => rama principal).

git status: Nos muestra el estado de los archivos, si están en rojo están untrakect y si están en verde están trakect.

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 (archivo): Es para subir los cambios hechos al espacio de trabajo local y poner un archivo en trakect. Podemos usar git add (archivo) o git add . para agregar y cambiar el estado de todos los archivos.

git commit -m "(mensaje)": Esto es para subir los archivos QUE ESTAN TRAKECT al repositorio local (la base de datos).

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

git log: Nos permite ver los commits que hemos hecho listándolos del más nuevo al más viejo, también les da una etiqueta a los commits para identificarlos y trabajar con ellos llamada sha (el conjunto de números y letras en verde manzana).

git log --oneline: permite mostrar los commits hechos sin tanta información.

git log --all --graph: permite mostrar los commits en un gráfico que muestra también la creación de ramas.

git log --all --graph --decorate --oneline: es una fusión entre git log --oneline y git log --all --graph.

git show: nos muestra los cambios que subimos en el último commit.

git diff (sha antiguo) (sha nuevo): muestras las diferencias entre un commit viejo y otro nuevo.

git reset (sha) --hard: nos permite "borrar" los commit que están por arriba de la sha especificado, "--hard" indica que va a borrar hasta el estado del archivo, es decir, lo va a dejar untracket y el archivo va a tener los cambios del último commit.

git reset (sha) --soft: hace lo mismo que --hard solo que esta vez el archivo se queda en el área de trabajo, es decir, queda tracket y no cambia su contenido.

git checkout (sha) (archivo): permite volver a ver el contenido de un commit antiguo sin borrar los otros commits y también se puede hacer un nuevo cambio, agregarlo al área de trabajo y subirlo al repositorio local, partiendo del commit donde ahora nos encontramos.

git checkout master (archivo): permite volver al commit actual y ver su contenido. "master" hace referencia a "(HEAD -> master)" que aparece, cuando introducimos el comando "git log" o "git log --oneline", en el commit actual.

git branch (nombre): esto nos permite crear una rama.

git branch --list: mostramos las ramas que tenemos. La rama en la que estamos esta de color verde.

git checkout (branch): para poder entrar a una rama.

git commit -am "(mensaje)": de esta manera no tenemos que escribir "git add" y después "git commit -m", sino que de una vez realizamos estos dos pasos con este comando.

git merge (branch): esto es para fusionar ramas. Debemos pararnos en la rama que queremos que absorba a la otra, ej.: si estamos parados en la rama B, y queremos que la rama X se fusione con la B, debemos pararnos en X y aplicar el comando.

git restore (archivo): permite que el archivo vuelva a estar como estuvo en el último commit.

git remote add (nombre de la conexión, siempre tratemos de llamarla origin) (url del repo remoto): este comando es para crear una conexión entre nuestro repo local y nuestro repo remoto (GitHub).

git remote -v: es para ver las conexiones que tenemos entre nuestros repositorios.

git remote set-url origin (url): se utiliza para cambiar la dirección url con las que conectamos los repos.

git pull origin master --allow-unrelated-histories: para traer los archivos del repo remoto al repo local (este se utiliza solo una vez cuando creamos el repositorio remoto).

git push origin master: para empujar nuestros archivos (si el repo remoto es nuevo) y cambios al repo remoto.

git pull origin master: para traer los cambios al repo local.

alias (nombre)="(comando)": sirve para guardar comandos en variables las cuales después podemos llamar, de esta manera no hace falta escribir el comando sino solo escribir el nombre de su variable, su alias.

git tag -a (version) -m "(mensaje)" (sha): de esta manera creamos tags.

git tag: para ver los tags que tenemos.

git show-ref --tags: para ver con que commit está enlazado ese tag.

git push origin --tags: para enviar los tags al repo remoto.

git tag -d (nombre tag): para borrar tags en el repo local.

git push origin :refs/tags/(nombre tag): para borrar tags en el repo remoto.

gitk: abre una ventana donde podemos ver los cambios que hemos hecho hasta ahora junto con la gráfica.

///////////////////////////////////////////////////////////////////////////////////////////////////////////

Git Bash comandos:

pwd --> muestra la ruta actual.

ls --> Lista los elementos (archivos y/o carpetas) que hay en la ruta actual.

ls -al --> Lista los elementos (archivos y/o carpetas) que hay en la ruta actual, junto con los elementos ocultos.

ls -al No es un comando, ls si lo es, pero -al son dos letras que hacen referencias, "a" (all muestra todo hasta los archivos ocultos) y "l" (coloca todos los archivos y elementos uno abajo de otro).

Cuando queremos usar letras que hacen cierta acción debemos poner -(letra), si queremos usar una palabra que hacen cierta acción debemos poner --(palabra).

clear --> Sirve para limpiar el bash (pantalla). Para este comando hay un atajo que es CTRL+L.

cd .. --> Nos permite retroceder una carpeta.

cd (nombre de la carpeta) --> Entro a una de las carpetas en la ruta actual. Cuando aparece ~ significa que estamos en la ruta principal, en este caso Ana. Si colocamos las primeras letras del nombre de la carpeta y presionamos TAB nos va a auto completar el nombre, en caso de que solo haya una carpeta con esas letras iniciales, o nos va lanzar carpetas que empiecen con esas letras, en caso de que existan carpetas con esas letras.

mkdir (nombre carpeta) --> crear carpeta.

touch (nombre.txt) --> crear archivo de texto plano.

cat (nombre del archivo) --> Ver archivo por medio de la terminal.

history --> permite ver el historial de comando usados.

! numero de linea --> Tomando en cuenta el historial de comandos podemos volver a ejecutar comandos con este comando, donde colocamos “!” y luego el número de linea que tiene el comando en el historial de comandos.

rm (nombre del archivo) --> sirve para borrar archivos. Importante, no aplicar este comando en el disco C ya que puede borrar nuestro servidor.

comando (rm, ls, etc..) --help --> nos dice todas las letras o palabras que se pueden usar para este comando.

code (archivo) --> nos permite crear un archivo y abrirlo en VSCode. Si ya está creado nos permite abrirlo.

ssh-keygen -t rsa -b 4096 -C "name\_email@gmail.com": este comando se ejecuta solo una vez para un equipo y se ejecuta en la carpeta principal del equipo (sabemos que estamos en ella porque aparece esto ~ después de MINGW64), sirve para crear una llave privada y pública para trabajar con el protocolo de seguridad ssh en GitHub.

eval $(ssh-agent -s): nos permite ver si nuestro puerto está activo para recibir archivos por ssh (si aparece Agent pid numero, significa que está funcionando).

ssh-add ~/.ssh/id\_rsa : para adherir nuestra llave privada al nuestro correo.

git clone (url repo remoto): para clonar un repositorio en otro equipo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**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

\*\*\*\*\*Usaremos el comando git branch para crear ramas dentro del proyecto, movernos entre ramas con checkout, fusionarlas con **merge**, así como trabajar con ramas de repositorios remotos\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

a) Git pull

b) Git clone

c) Git download

d) Git remote

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Cómo hacer tu primer pull request en GitHub

[**Fernando Cardellino**](https://www.freecodecamp.org/espanol/news/author/fernando-cardellino/)

## ¿Qué es hacer un forking?

Cuando nos gusta el repositorio de alguien y nos gustaría tenerlo en nuestra cuenta de GitHub, hacemos un fork o bifurcación para poder trabajar con él en forma separada.

Cuando hacemos un fork de un repositorio, obtenemos una instancia de todo el repositorio con todo su historial. Luego, podemos hacer lo que queramos sin afectar la versión original.

## ¿Qué es un pull request?

Los pull requests son la forma de contribuir a un proyecto grupal o de código abierto.

Por ejemplo, un usuario llamado Harry realiza un fork de un repositorio de ThanoshanMV y le efectúa algunos cambios. Ahora Harry puede hacer un pull request a ThanoshanMV, pero dependerá de ThanoshanMV aceptar o declinarlo. Es como decir: "ThanoshanMV, ¿podrías por favor extraer (pull) mis cambios?"

## ¿Qué significa contribuir?

No solo podemos contribuir a un proyecto de código abierto con código, sino que podemos hacerlo de muchas otras maneras. Algunas de ellas están descritas abajo.

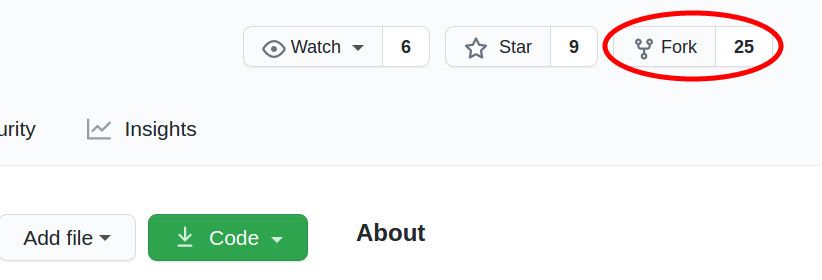
Como indica la guía de hacktitude para iniciados de la firma IT [99xtechnology](https://99x.io/), podemos contribuir a un proyecto de código abierto de las siguientes maneras:

1. Diseñando: Puedes construir los diseños de un proyecto para mejorar su usabilidad, mejorar la navegación y el menú del proyecto en función de lo indicado por investigaciones acerca de las preferencias del usuario, crear arte para logotipos o camisetas y proporcionar guías de estilo para el proyecto.
2. Escribiendo: Puedes escribir y mejorar la documentación del proyecto o traducirla, iniciar un boletín o escribir tutoriales para el proyecto y tomar nota de los aspectos más destacados de la lista de correo, u organizar una carpeta de ejemplos mostrando cómo se utilizan los proyectos.
3. Organizando: Puedes vincular problemas duplicados, sugerir nuevas etiquetas de problemas, sugerir cerrar viejos problemas aún abiertos y hacer preguntas sobre problemas abiertos recientemente para hacer avanzar la discusión.
4. Ayuda a otros: Responde preguntas de problemas aún en discusión, revisa el código enviado por otras personas y ofrécete a ser mentor de otro colaborador.
5. Elaborando código: Ayuda a resolver problemas aún en discusión, consulta si puedes proveer nuevas funciones y mejorar las herramientas y las pruebas.

## ¡Realicemos nuestro primer pull request!

### 1. Realicemos un fork del repositorio

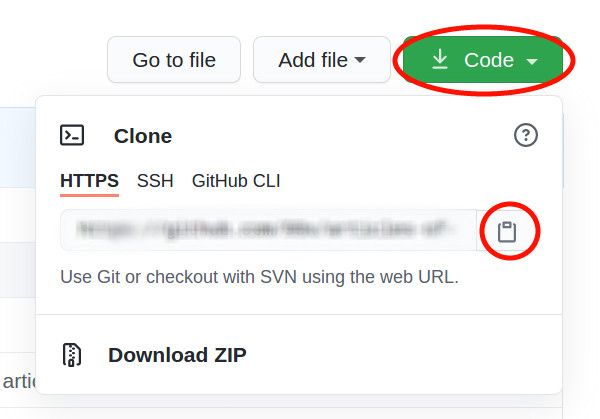
Realiza un fork del repositorio haciendo un clic en el botón fork de la parte superior de la página. Esto creará una instancia del repositorio completo en tu cuenta.



### 2. Clona el repositorio

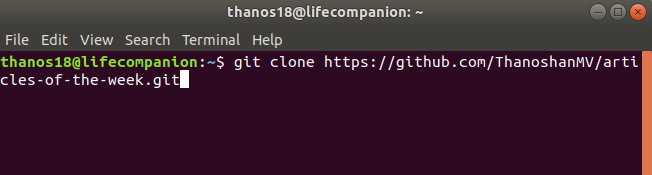
Una vez que el repositorio esté en tu cuenta, clónalo a tu computador para trabajarlo localmente.

Para clonarlo, has clic en el botón "Code" y copia el link.



Abre la terminal y ejecuta el siguiente comando. Esto clonará el repositorio localmente.

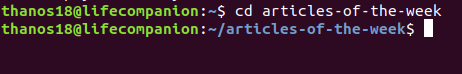
$ git clone [DIRECCIÓN HTTPS]



Ahora hemos configurado una copia de la rama maestra desde el repositorio principal del proyecto en línea.

Debemos ir al repositorio clonado ejecutando el siguiente comando:

$ cd [NOMBRE DEL REPOSITORIO]



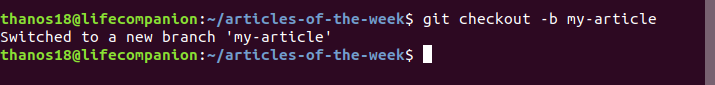
### 3. Crea una rama

Es una buena práctica crear una rama (branch) nueva cuando trabajas con repositorios, ya sea que se trate de un proyecto pequeño o estés contribuyendo en un equipo de trabajo.

El nombre de la rama debe ser breve y debe reflejar el trabajo que estamos haciendo.

Ahora crea una rama usando el comando git checkout:

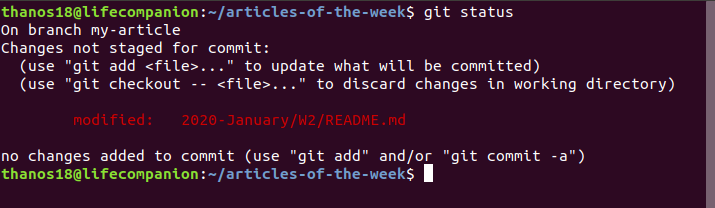
$ git checkout -b [Nombre de la Rama]



### 4. Realiza cambios y confírmalos

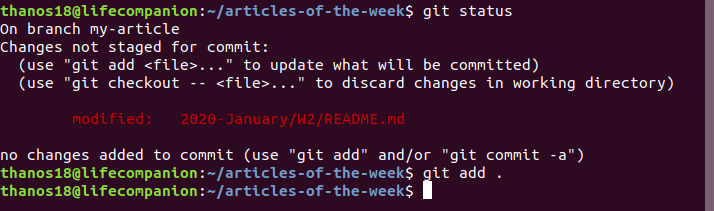
Has cambios esenciales al proyecto y guárdalos.

Luego ejecuta git status , y verás los cambios.



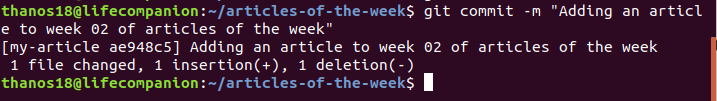
Agrega esos cambios a la rama recién creada usando el comando git add:

$ git add .



Ahora confirma esos cambios utilizando el comando git commit:

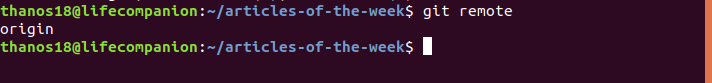
$ git commit -m "Adding an article to week 02 of articles of the week"



### 5. Envía los cambios a GitHub

Para enviar los cambios a GitHub, debemos identificar el nombre del repositorio remoto.

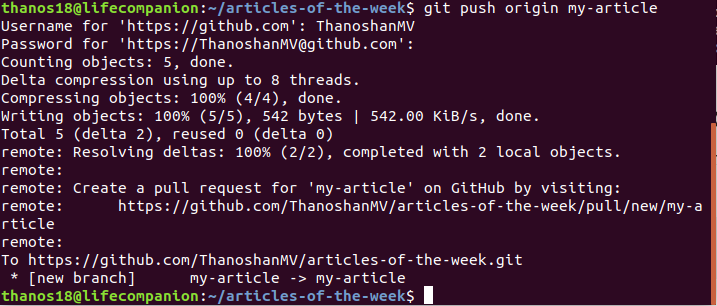
$ git remote



Para este repositorio el nombre es "origin".

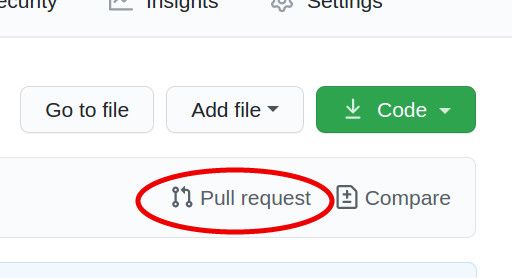
Luego de identificar el nombre podemos enviar en forma segura los cambios a GitHub.

git push origin [Nombre de la Rama]



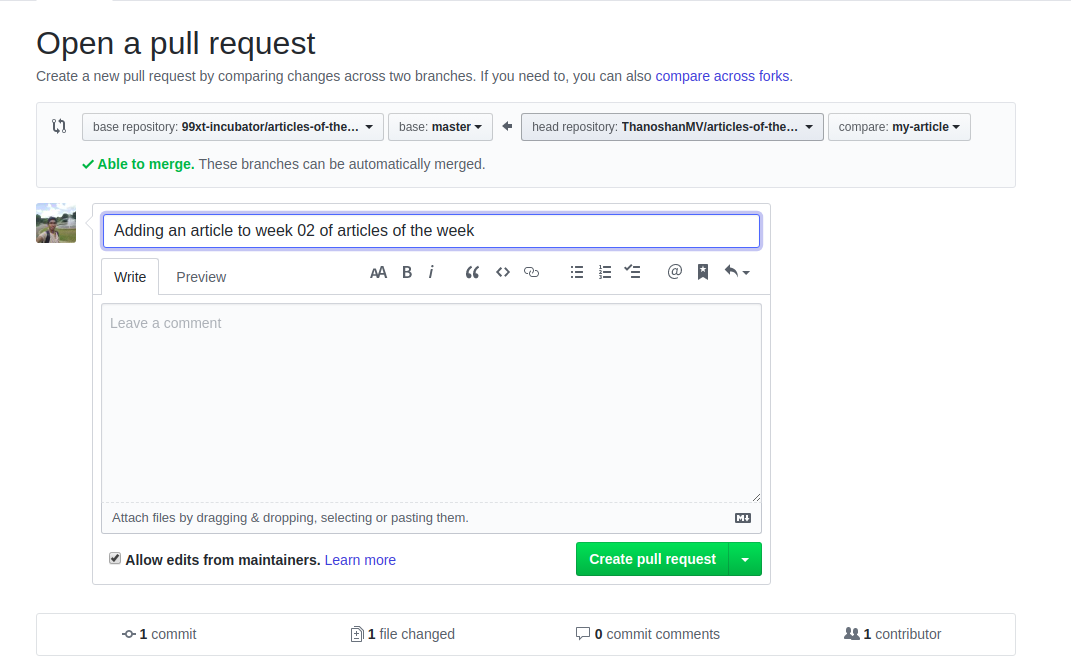
### 6. Crea un pull request

Ve a tu repositorio en GitHub y verás un botón llamado "Pull request", has clic en él.



Por favor, provee todos los detalles necesarios de lo que has hecho (puedes referenciar problemas utilizando "#"). Ahora, envía el pull request.

¡Felicitaciones! Has hecho tu primer pull request.



Si tu pull request es aceptado recibirás un mail.

### 7. Sincroniza tu rama maestra con la del repositorio original

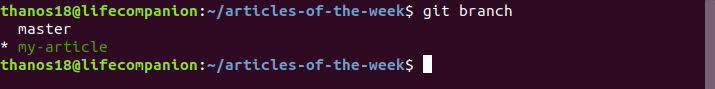
Antes de enviar cualquier pull request al repositorio original debes sincronizar tu repositorio con aquel.

Incluso si no vas a enviar un pull request al repositorio original, es mejor efectuar la sincronización, ya que pueden haberse agregado algunas prestaciones o funciones adicionales y haberse corregido algunos errores desde la vez que realizaste un fork de aquel repositorio.

Sigue estos pasos para actualizar/sincronizar aquellos cambios con tu rama maestra:

1. Primero, revisa en que rama estás ubicado.

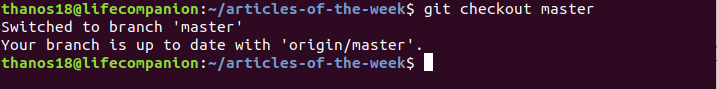
$ git branch



Esto enumerará todas las ramas y indicará en verde la rama actual o activa.

2. Cambia a la rama maestra.

$ git checkout master

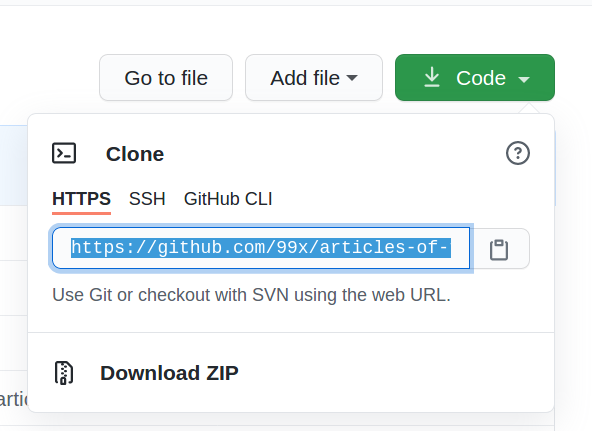


3. Agrega el repositorio original como un repositorio upstream.

Para poder extraer los cambios desde el repositorio original a tu versión local, necesitas agregar el repositorio Git original como un repositorio upstream.

$ git remote add upstream [HTTPS]

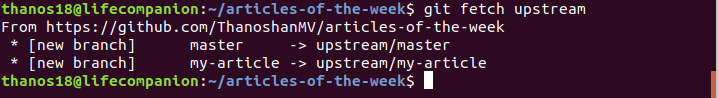
Aquí, [HTTPS] es el URL que debes copiar del repositorio del propietario.

remote-add

4. Busca (fetch) el repositorio.

Busca todos los cambios  del repositorio original. Las confirmaciones (commits) del repositorio original serán almacenadas en una rama local llamada upstream/master.

$ git fetch upstream



5. Fusionala.

Fusiona los cambios de la rama upstream/master a tu rama maestra local. Esto hará que tu rama maestra se sincronice con el repositorio upstream sin perder tus cambios locales.

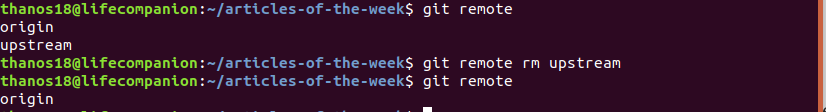
$ git merge upstream/master

6. Envía (push) los cambios a GitHub

En este punto tu rama local está sincronizada con la rama maestra del repositorio original. Si deseas actualizar el repositorio de GitHub, necesitas enviar tus cambios.

$ git push origin master

**NOTA:** Luego de sincronizar tu rama maestra puedes eliminar el repositorio upstream, si lo desea. Pero lo necesitará para actualizar/sincronizar tu repositorio en el futuro, por lo que es una buena práctica conservarlo.

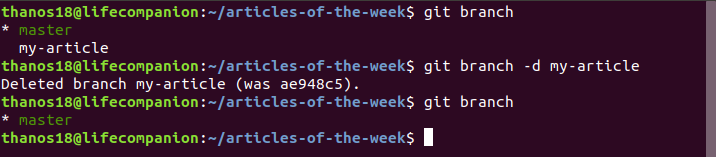


$ git remote rm [Nombre del Repositorio Remoto]

### 8. Elimina ramas innecesarias

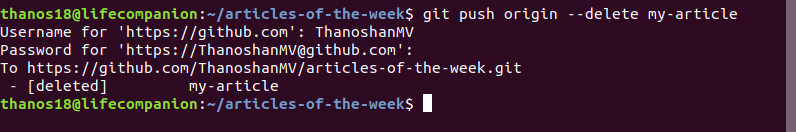
Las ramas son creadas para propósitos especiales. Una vez que ese propósito se cumple, aquellas ramas ya no son necesarias, por lo que puedes eliminarlas.

$ git branch -d [Nombre de la Rama]



También, puedes eliminar su versión en GitHub.

git push origin --delete [Nombre de la Rama]



## Conclusión

GitHub es una poderosa herramienta para controlar el historial de versiones. Todos pueden contribuir a proyectos de código abierto mediante pull requests. Las contribuciones no siempre son un código; también hay otras formas de contribuir.

Finalmente, debo decirte que no debes preocuparte si tus pull requests son rechazadas. Los encargados de los proyectos dedican mucho tiempo a mejorarlos y saben mucho más sobre sus proyectos que nosotros. Así que no te preocupes si tu solicitud no es tenida en cuenta.

Mantente fuerte, positivo y nunca te rindas.  
― Roy T. Bennett, [The Light in the Heart](https://www.goodreads.com/work/quotes/49604402)

Este artículo fue originalmente publicado en [Medium](https://medium.com/@mvthanoshan9/how-to-make-your-first-pull-request-on-github-9aefca5cc837).

Puedes contactarme y conectar conmigo en [Twitter](https://twitter.com/ThanoshanMV).

**¡Sigue contribuyendo al mundo del código abierto!**

Traducido del arículo de [**Thanoshan MV**](https://www.freecodecamp.org/news/author/thanoshan/)**-**[**How to make your first pull request on GitHu**](https://www.freecodecamp.org/news/how-to-make-your-first-pull-request-on-github-3/)