



# Uso Básico de Git

## Guía práctica

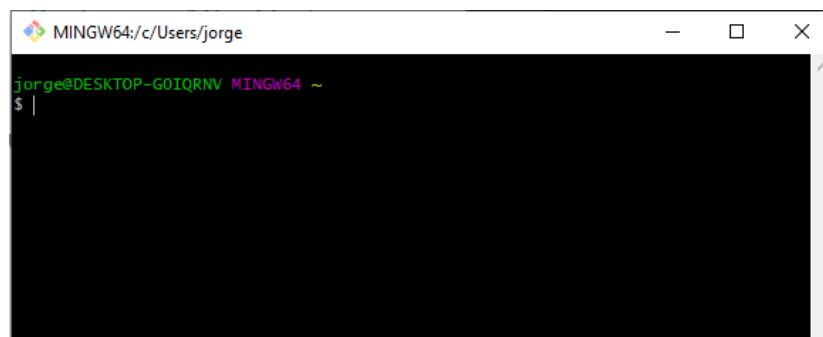
## ¿Qué es Git?

Es un sistema de control de versiones, es distribuido, es decir que múltiples personas pueden trabajar en equipo, es open source y también se adapta a todo tipo de proyectos desde pequeños hasta grandes, además, se pueden fusionar archivos, guarda una línea de tiempo a lo largo de todo el proyecto.

Para comenzar a utilizar debe descargar la aplicación desde la siguiente dirección: <https://git-scm.com/downloads>



Una vez realizada la instalación debe abrir la consola de git haciendo clic en git bash.

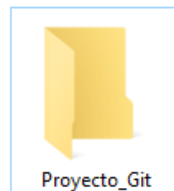


Aquí algunos de los comandos básicos de git, que son necesarios para poder ocupar nuestro software de control de Versiones.

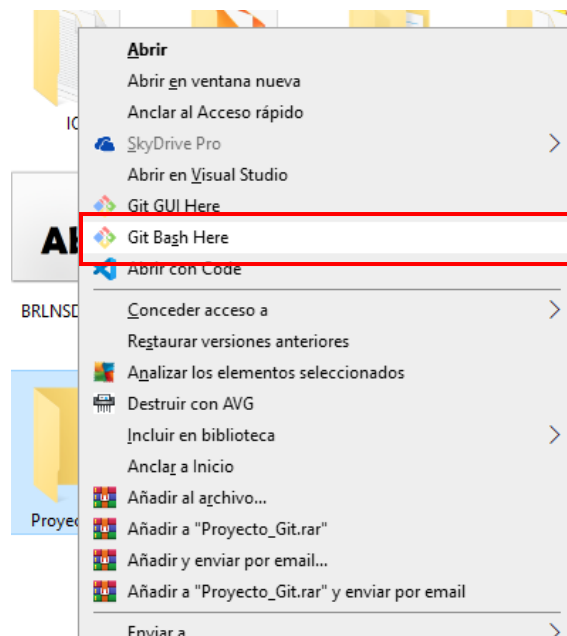
```
// Conocer la versión de git instalada  
git version
```

```
// Ayuda sobre los comandos  
git help
```

Crearemos una carpeta donde dejaremos nuestro proyecto de programación, para este ejemplo se creara en mi disco D:, con el nombre de Proyecto\_Git



Seleccionamos la carpeta y con el botón derecho seleccionamos la opción Git Bash



Se abrirá la consola de **git** en nuestra carpeta y ahora le indicaremos que esta será la carpeta de nuestro repositorio.

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git
```

Iniciamos el repositorio en nuestra carpeta donde tendremos nuestro proyecto de programación

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git
$ git init
Initialized empty Git repository in D:/Proyecto_Git/.git/

jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ |
```

Además de crear nuestro repositorio creara una carpeta oculta con el nombre de **.git**, que permite vincular las opciones de versión de archivos para lo que necesitemos



Ahora revisaremos el estado de nuestros archivos de nuestra carpeta

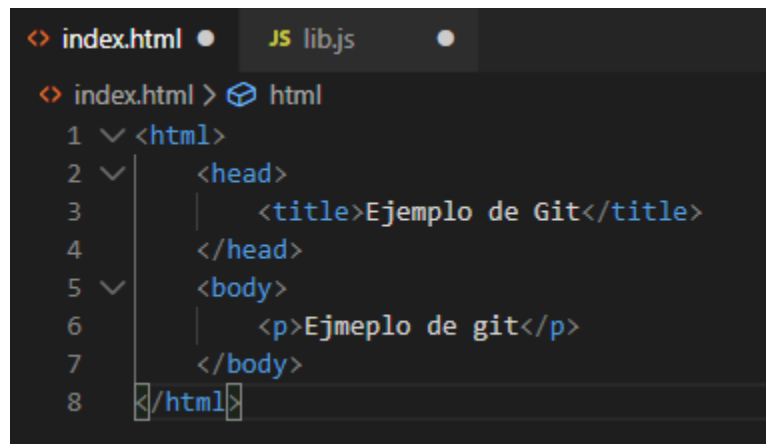
```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git status
On branch master

No commits yet

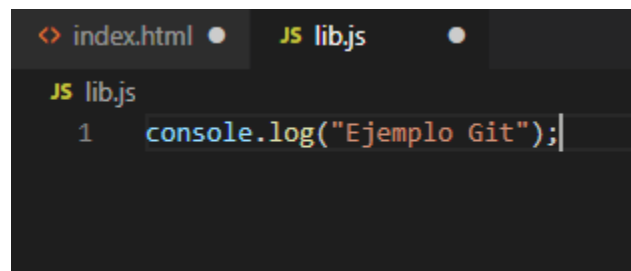
nothing to commit (create/copy files and use "git add" to track)
```

Al colocar el comando **git status**, y no haber ningún archivo, debería mostrar el resultado de la imagen anterior.

Crearemos dos archivos solo de manera de Prueba. [ index.html y lib.js ]

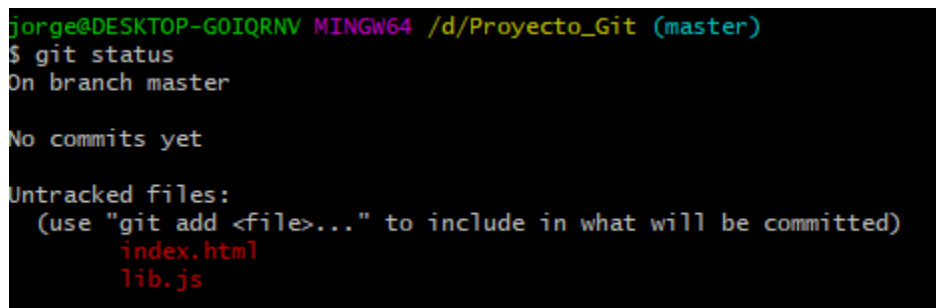


```
<> index.html • JS lib.js
<> index.html > html
1 <html>
2   <head>
3     <title>Ejemplo de Git</title>
4   </head>
5   <body>
6     <p>Ejemplo de git</p>
7   </body>
8 </html>
```



```
<> index.html • JS lib.js
JS lib.js
1 console.log("Ejemplo Git");
```

Ahora preguntaremos por el estado de nuestro repositorio **git status**



```
jorge@DESKTOP-GOIQRNV MINGW64 /d/Proyecto_Git (master)
$ git status
On branch master

No commits yet

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

Como pueden ver se encuentra en color rojo los dos archivos que creamos y deberemos agregarlo para que formen parte del control de gestión de cambio, podemos hacerlo uno a uno, haciendo referencia al nombre y la extensión del archivo o podremos hacerlo incorporando todos los archivos.

// **git add .**

Agregar todos los archivos para que esté pendiente de los cambios

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git add index.html

jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git add .
```

Revisamos nuevamente el estado de nuestro repositorio y deben estar incorporado los archivos.

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
        new file:   lib.js
```

Ahora lo que debemos realizar es crear el punto de partida de la primera versión o fotografía del proyecto en este momento, para ello debemos utilizar el comando **git commit**

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git commit -m "Inicio de Nuestro Proyecto de Git"
[master (root-commit) 1b8af43] Inicio de Nuestro Proyecto de Git
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
 create mode 100644 lib.js
```

Todo comentario debe ir representado con una descripción representativa de la actividad que se realizó en el proyecto o los cambios en los archivos.

Para poder visualizar la lista de cambios que ha tenido nuestro proyecto desde el más antiguo al más reciente lo debemos hacer con **git log**.

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)
$ git log
commit 1b8af434948fb2e4e0e5def6a95c27b7fa810a82 (HEAD -> master)
Author: jcortes <jorge.cortes.gallardo@gmail.com>
Date:   Mon Apr 27 22:21:29 2020 -0400

    Inicio de Nuestro Proyecto de Git
```



## Los Tres Estados o Flujo de Trabajo de Git

Presta atención. Esto es lo más importante que debes recordar acerca de Git si quieres que el resto de tu proceso de aprendizaje prosiga sin problemas.

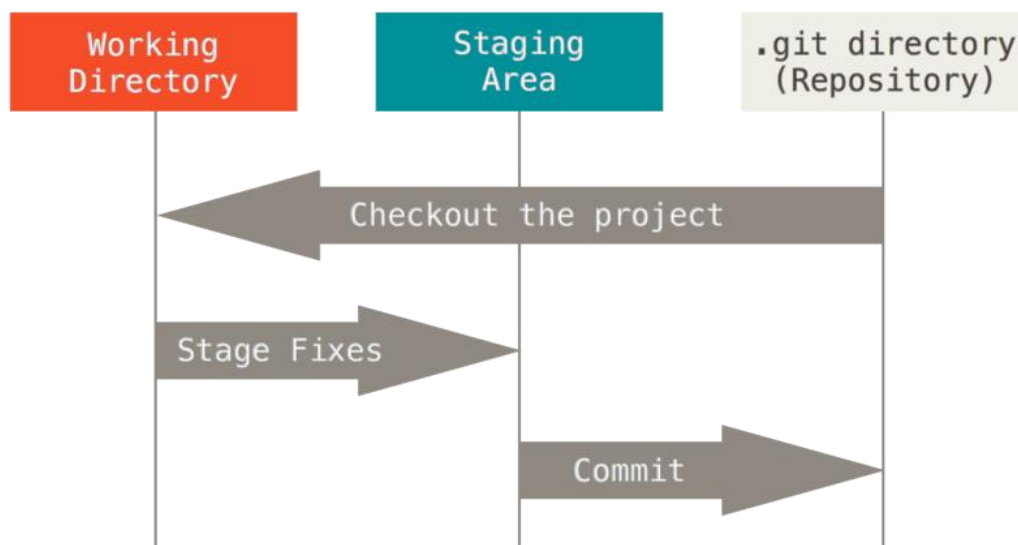
Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged).

**Confirmado:** significa que los datos están almacenados de manera segura en tu base de datos local.

**Modificado:** significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.

**Preparado:** significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: El directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).



El directorio de Git es donde se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro computador.

El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

El área de preparación es un archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice (“index”), pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

El flujo de trabajo básico en Git es algo así:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

En palabras más simples, cuando realizamos cambios en nuestros archivos, el comando `status` verificará que archivos han sido modificados. Cuando deseemos registrar esos cambios tendremos que agregarlos con `add`. así ya estará listo para poder hacer un `commit`. El `commit` realiza la copia de ese instante para poder volver en el tiempo si es que es necesario.

Antes de Seguir algunos comandos que nos servirán ya que nos permiten tener un mayor control y más claro del destalle de nuestro trabajo en el repositorio.

```
// Muestra en una línea los commit realizados
git log --oneline
```

```
// Muestra en una línea los commit realizados, pero más elegante
git log --oneline --decorate --all --graph
```

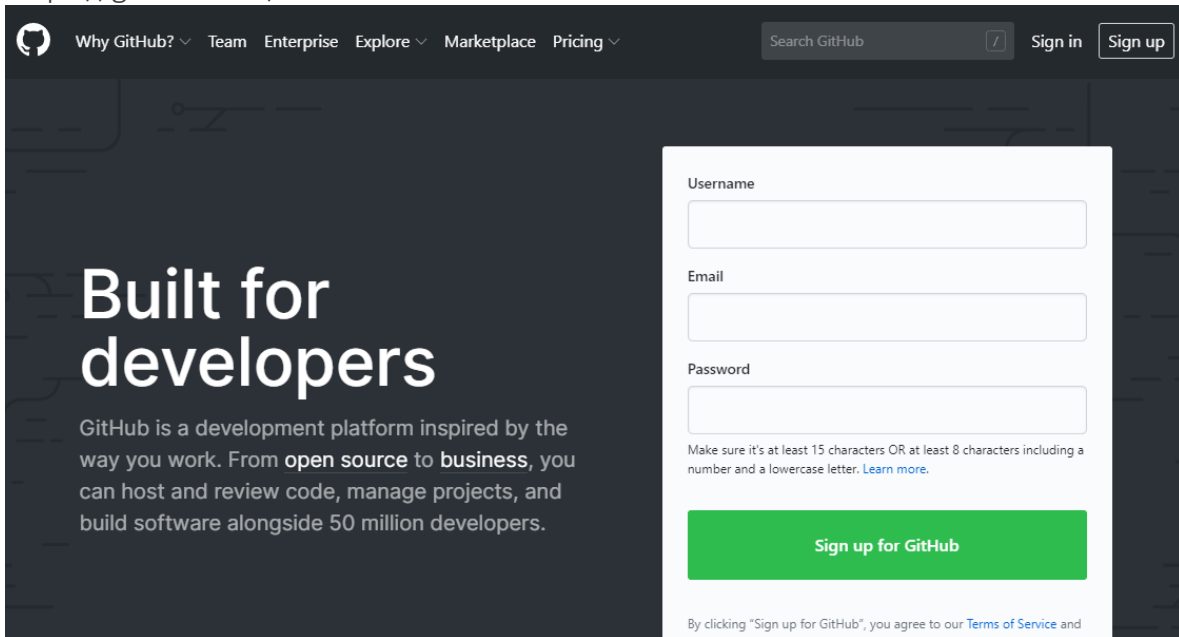


```
// Solo muestra los archivos modificados  
git status -s
```

## ¿Qué es Github?

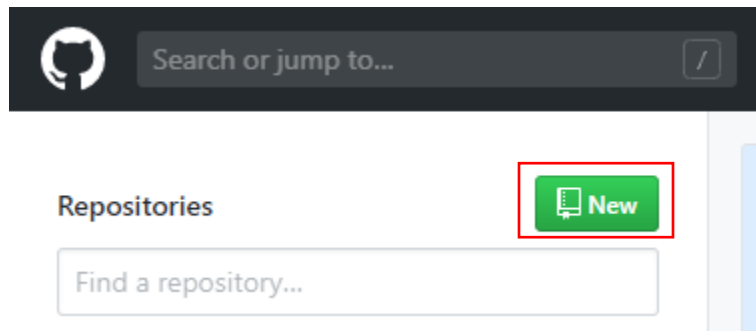
Es una plataforma de desarrollo colaborativo, o también llamada la red social de los desarrolladores donde se alojan los repositorios, el código se almacena de forma pública, pero se puede hacer privado con una cuenta de pago.

<https://github.com/>



The screenshot shows the GitHub homepage with a dark theme. On the left, the text "Built for developers" is prominent, followed by a description of GitHub as a development platform. On the right, there is a white sign-up form with fields for Username, Email, and Password. Below the Password field, there is a note about password requirements and a link to "Learn more". A green button labeled "Sign up for GitHub" is at the bottom of the form. The top navigation bar includes links for "Why GitHub?", "Team", "Enterprise", "Explore", "Marketplace", and "Pricing", along with a search bar and "Sign in" and "Sign up" buttons.

En su dashboard deben seleccionar la opción NEW para crear su repositorio

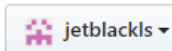


Y llenan el formulario como se muestra en el siguiente ejemplo:

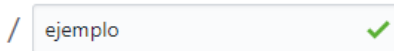
## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

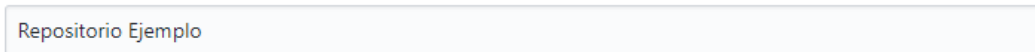


Repository name \*



Great repository names are short and memorable. Need inspiration? How about [refactored-octo-eureka?](#)

Description (optional)



☒  **Public**

Anyone can see this repository. You choose who can commit.

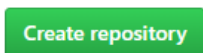
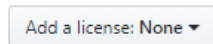
☐  **Private**

You choose who can see and commit to this repository.

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

☐ **Initialize this repository with a README**

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



Mostrará el área de configuración del repositorio y como se podrán dar cuenta en la primera parte GitHub no indica como poder crear el repositorio que son los pasos y comandos que realizamos anteriormente, entonces en este caso debemos preocuparnos de la según opción:

...or push an existing repository from the command line

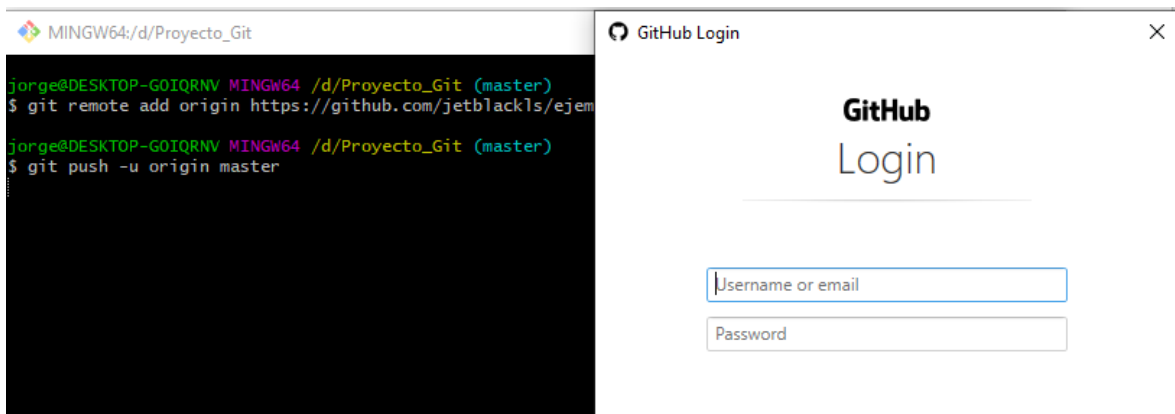
```
git remote add origin https://github.com/jetblackls/ejemplo.git  
git push -u origin master
```



Y lo pegamos en nuestro git bash

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)  
$ git remote add origin https://github.com/jetblackls/ejemplo.git
```

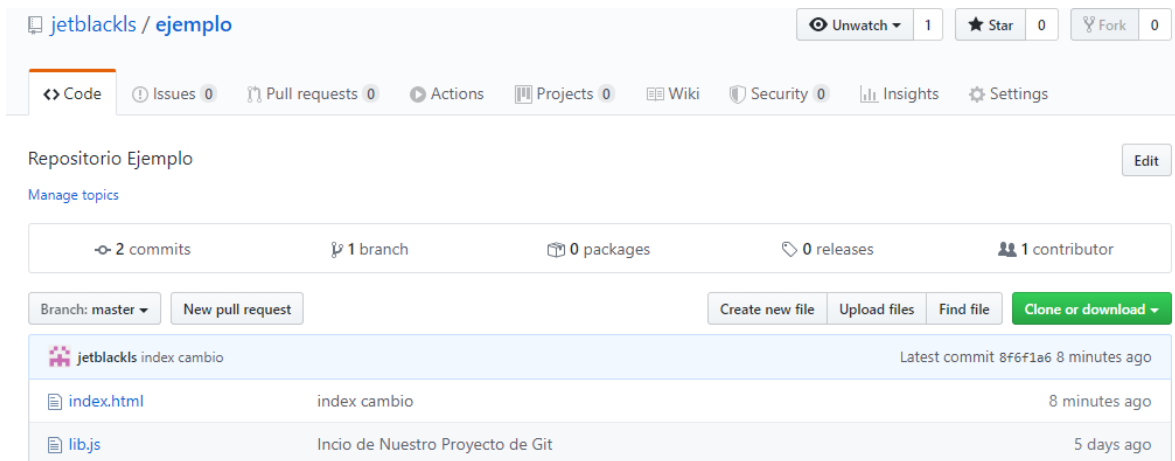
Ahora hacemos el Push, que significa enviar todos nuestros archivos del ultimo commit a nuestro repositorio en la nube, por lo que nos pedirá autenticarnos con las credenciales de nuestra sesión en GitHub.



Una vez ingresada las credenciales comenzara a realizar el envío de nuestro repositorio local hacia el repositorio creado en GitHub

```
jorge@DESKTOP-G0IQRNV MINGW64 /d/Proyecto_Git (master)  
$ git push -u origin master  
Enumerating objects: 6, done.  
Counting objects: 100% (6/6), done.  
Delta compression using up to 4 threads  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (6/6), 505 bytes | 252.00 KiB/s, done.  
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/jetblackls/ejemplo.git  
 * [new branch]      master -> master  
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Ahora hacemos clic en la pestaña code y deberíamos ver nuestros archivos



Hasta aquí estaría lo más básico e importante de lo que se necesita saber, pero dejare algunas cosas que son necesarias como lo son los Tags que sirven para hacer versiones.

```
// Crear un tags  
git tag versionAlpha -m "versión alpha"
```

```
// Listar tags  
git tag
```

```
// Borrar tags  
git tag -d nombreTags
```

```
// Mostrar información del tag  
git show nombreTag
```

```
// Hacer una versión en un commit anterior ej: f52f3da  
git tag -a nombreTag f52f3da -m "version alpha"
```

Y ahora para subir nuestros tags a GitHub

```
git push --tags
```



## ALGUNOS COMANDOS AVANZADOS

### [ Clonar repositorio ]

Para descargar un repositorio completo basta con tomar la url ej: <https://github.com/jetblackls/ejemplo> y ejecutar el siguiente comando en alguna carpeta de su computadora.

```
git clone https://github.com/bluuweb/tutorial-github.git nombreCarpeta
```

### [ Fetch ]

Este comando hace la comparación de nuestros archivos locales con los del servidor, si existiera alguna diferencia nos pediría realizar un get pull para realizar un match de nuestros archivos locales.

```
git fetch
```

## [ Viajes a través de los commit ]

Vamos a conocer como podemos movernos entre los diferentes commit que tengamos registrados, supongamos que tenemos los siguientes commit:

f82f457 (HEAD -> master) más comandos agregados  
f52f3da nuevos comandos en fundamentos.md  
e4ab8af mi primer commit

```
// Viajamos al commit en específico f52f3da  
git reset --mixed f52f3da
```

```
// Viajamos al commit en específico f52f3da y eliminamos los cambios futuros  
git reset --hard f52f3da
```

```
// Muestra todos los cambios incluso si borramos los commit  
git reflog
```

```
// Viajamos al commit en específico f52f3da y podemos restaurar los archivos  
git reset --hard f52f3da
```

Si no hicimos un commit pero aun así queremos revertir los cambios en un archivo específico podríamos utilizar el siguiente comando:

```
git checkout -- nombreArchivo.conExtensión
```

Si deseamos destruir todos los cambios sin haber realizado un commit podemos utilizar:

```
git reset --hard
```

Para más detalles o más información les recomiendo visitar la página oficial de Git <https://git-scm.com/book/es/v2/Herramientas-de-Git-Reiniciar-Desmitificado>.

## [ Ignorando Archivos ]

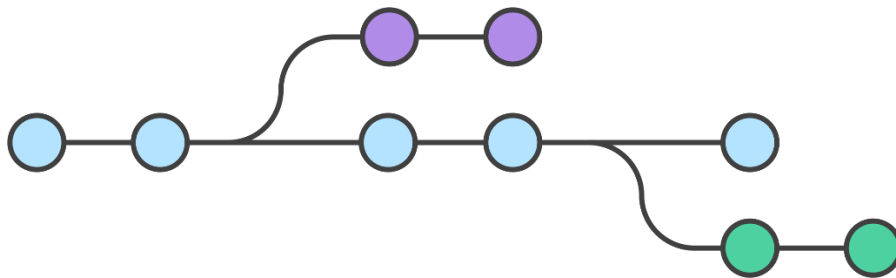
Para no hacer seguimiento de carpetas o archivos, debemos crear el siguiente archivo:

.gitignore

Y dentro del incorporar todo aquello que no queramos que sea parte de la actualización en nuestro push, un ejemplo sería así:

```
archivo.js // Ignora el archivo en cuestion  
*.js // Ignora todos los arhivos con extensión .js  
node_modules/ //Ignora toda la carpeta
```

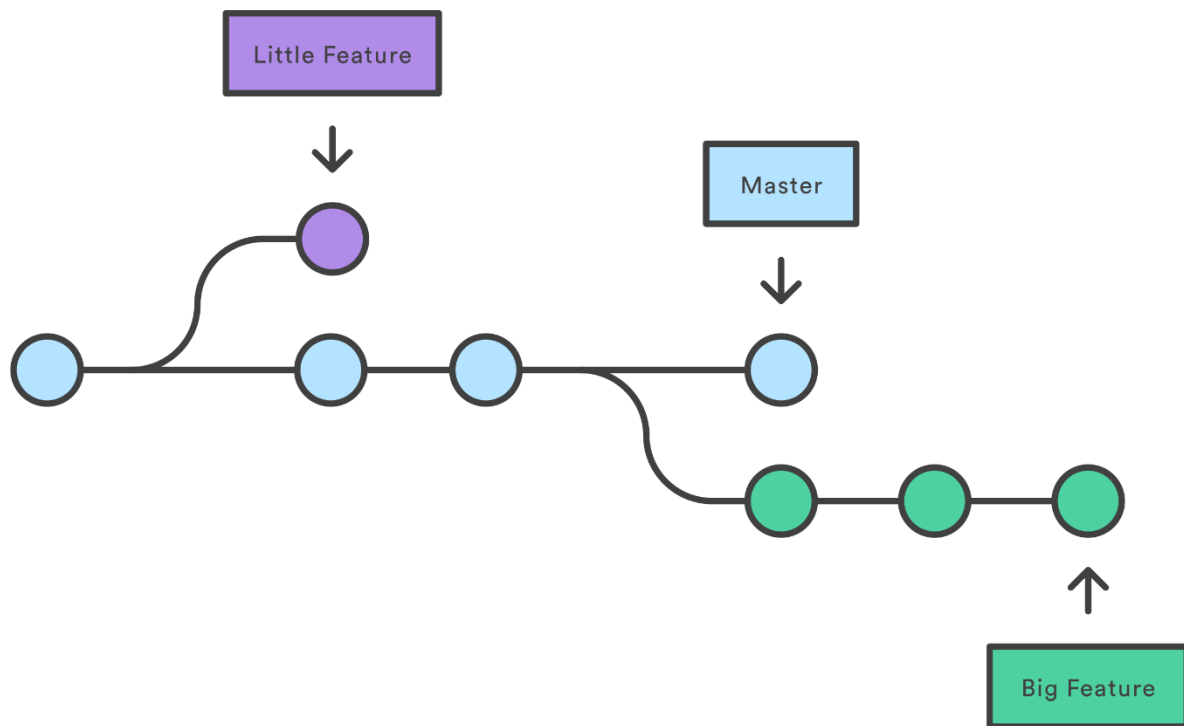
## [ Ramas o Branch ]



La creación de ramas en otros sistemas de control de versiones puede abarcar mucho tiempo y espacio de disco. En Git, las ramas son parte del proceso de desarrollo diario.

Las ramas de Git son un puntero eficaz a las instantáneas de tus cambios. Cuando quieres añadir una nueva función o solucionar un error (sin importar su tamaño), generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código se fusione con el código base principal, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal.

La rama por defecto es **Master**



Para mostrar o enumerar todas las ramas de tu repositorio.

```
git Branch  
// o  
git branch -list
```

Crea una nueva rama llamada <penka>. Este comando *no* extrae la nueva rama.

```
git branch penka
```

Elimina la rama especificada. Esta es una operación segura, ya que Git evita que elimines la rama si tiene cambios que aún no se han fusionado.

```
git branch -d penka
```



Fuerza la eliminación de la rama especificada, incluso si tiene cambios sin fusionar. Este comando lo puedes usar si quieres eliminar de forma permanente todas las confirmaciones asociadas con una línea concreta de desarrollo.

```
git branch -D penka
```

```
// Nos movemos a la nueva rama  
git checkout penka
```

Unir nuestra rama Master con cualquier rama creada, debemos estar siempre posicionado en la rama Master.

```
git checkout master  
git merge penka
```