



SOFTFACTORY

Git y GitHub

Guía para principiantes

Brian Pérez López
brianpl990227@gmail.com



Contenido

¿Qué es Git y GitHub?	2
Instalación en Windows de Git	2
Nuestro primer repositorio y commit	5
Ir a Hitos	11
Uso del gitignore	14
Ramas y Uniones	15
GitHub	17
Conclusiones	22



¿Qué es Git y GitHub?

Git es un software de control de Versiones creado por Linus Torvals, el creador de Linux. Algunas de las ventajas que nos puede ofrecer son:

- Auditoria de código
- Control de cambios en el tiempo
- Uso de etiquetas para versionar
- Facilita en gran medida el trabajo en equipo



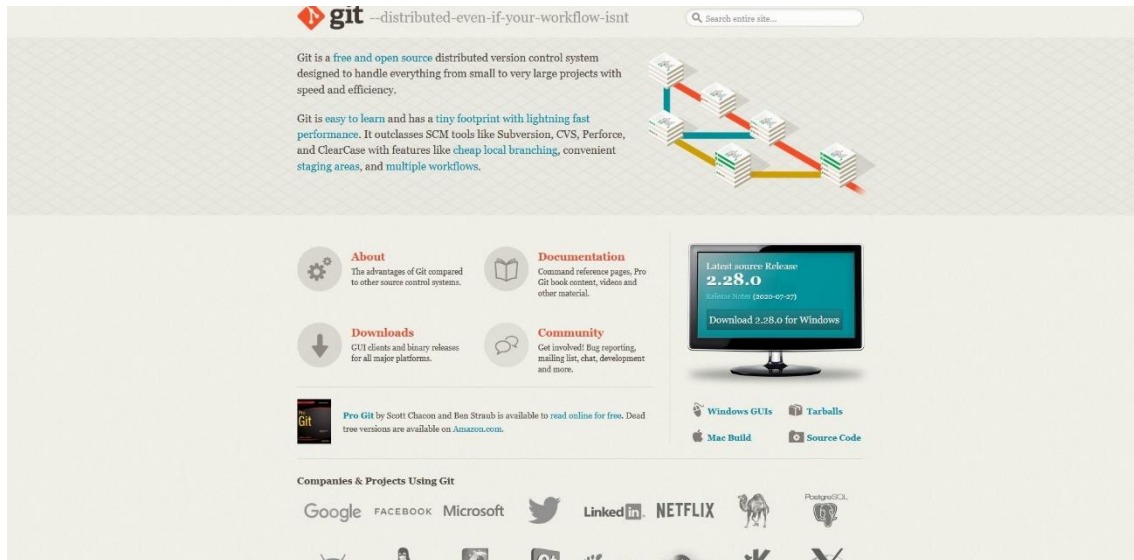
Pero también estoy seguro que has escuchado hablar de GitHub. Cabe aclarar que Git y GitHub no es lo mismo, Git es el sistema de control de versiones y GitHub es una red social que utiliza este sistema para nuestros proyectos con el uso de repositorios.



Instalación en Windows de Git

Git no solo está disponible para Windows, también lo está para Linux y MacOS, en este documento hablaremos sobre su descarga, instalación y puesta en marcha de Git.

Para instalar Git si usamos un sistema operativo Windows podemos hacerlo descargando el instalador desde su página oficial <https://git-scm.com/> una vez ya este dentro de la web aparecerá la siguiente interfaz y presionamos el botón que dice Download (versión actual) for Windows. Cuando presione en este botón la descarga de Git iniciará.



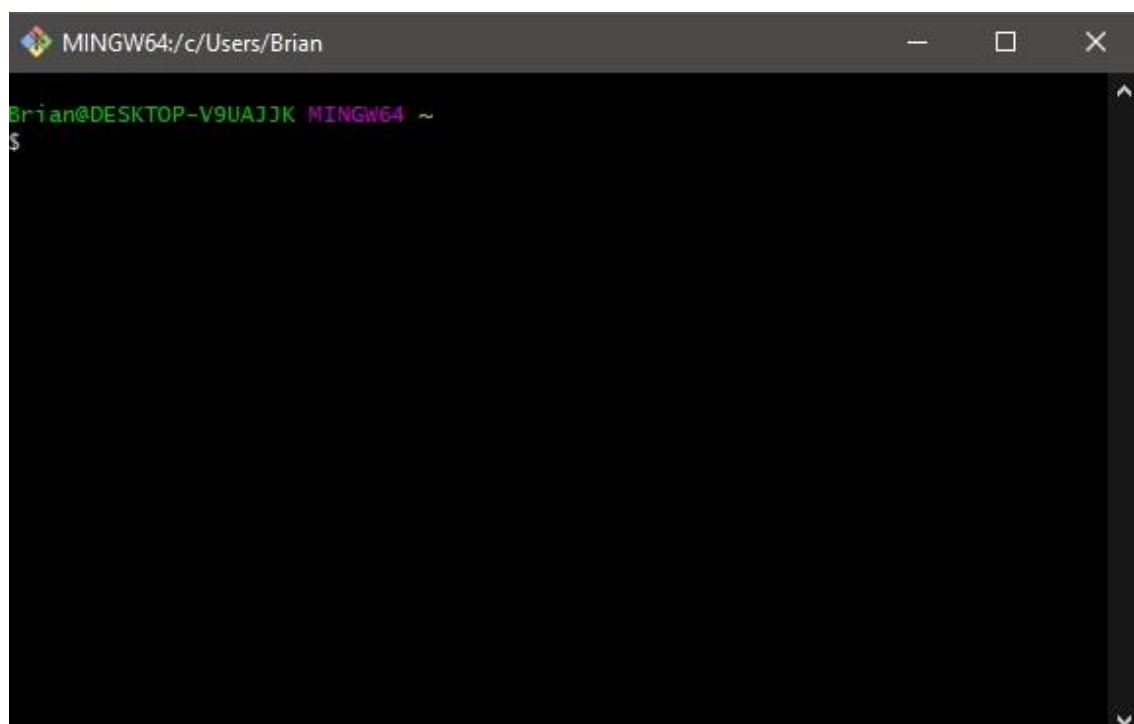
Una vez ya tengamos el instalador y lo ejecutemos llegaremos a esta vista donde dejaremos los valores por defectos



Después nos aparecerá con que IDE trabajamos, donde seleccionaremos el que usted usa, en mi caso Visual Studio Code.



A continuación, le damos a todos los valores por defectos que aparecen y dejamos que se instale. Una vez instalado escribimos en el menú inicio Git Bash, damos click y nos aparecerá la siguiente consola.





Para probar la versión que hemos instalado escribimos en esa consola:

```
git --version
```

Una vez escrito esto se muestra en dicha consola la versión de git que acabamos de instalar. No necesariamente hay que poner los comandos en esa consola, también se puede usar el CMD de Windows

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.18362.30]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Brian>git --version
git version 2.9.3.windows.1

C:\Users\Brian>
```

Nuestro primer repositorio y commit

Una vez ya todo este instalado, lo siguiente es decirle a Git quién eres tú, para que cuando se vaya a hacer algún cambio se sepa quien lo hizo en un proyecto con más desarrolladores para ello usamos el siguiente comando:

```
git config --global user.name "Brian"
```

```
git config --global user.email "brianpl990227@gmail.com"
```

Una vez le hayamos dicho a git quienes somos podemos verificarlo mediante

```
git config --global -l
```



A continuación, mostramos un ejemplo práctico con el cmd de Windows

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.18362.30]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

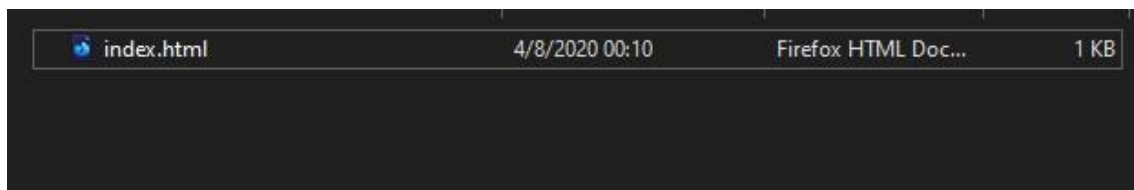
C:\Users\Brian>git config --global user.name "Brian"

C:\Users\Brian>git config --global user.email "brianp1990227@gmail.com"

C:\Users\Brian>git config --global -l
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=Brian
user.email=brianp1990227@gmail.com
core.autocrlf=true

C:\Users\Brian>
```

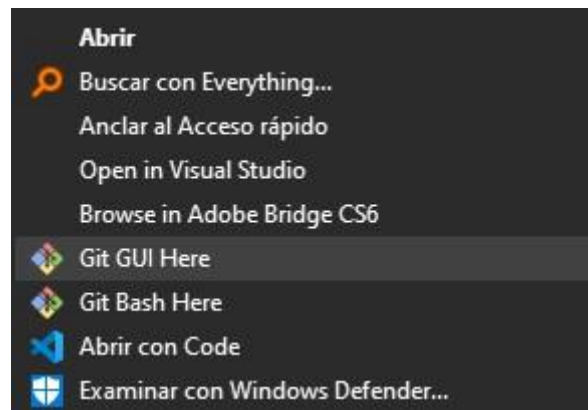
Ahora procederemos a crear un proyecto sencillo donde vamos a utilizar Git. Creamos una carpeta donde hicimos un html.



Y dentro de ese html un código sencillo al cual le aplicaremos un control de versiones.

```
C: > Users > Brian > Desktop > asd > index.html > ..
1  <!DOCTYPE html>
2  <html>
3  |   <head>
4  |       <title>Usando Git</title>
5  |   </head>
6  |   <body>
7  |       <h1>SoftFactory</h1>
8  |   </body>
9  </html>
```

Para aplicar el control de versiones damos click derecho a la carpeta raíz del proyecto en mi caso se llama asd (soy vago para dar nombres) y le damos Git Bash Here.



Esto nos llevará a la ruta de nuestro proyecto y aplicaremos el siguiente comando en la consola

`git init`

Esto nos creara una carpeta oculta llamada `.git` esta carpeta tiene las referencias necesarias para que el sistema de control de versiones funcione correctamente. Ahora para que git les haga un seguimiento a los archivos de nuestro proyecto, primero deberíamos saber el estado actual de estos archivos esto lo resolvemos con el comando

`git status`

```
MINGW64:/c/Users/Brian/Desktop/asd
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd
$ git init
Initialized empty Git repository in C:/Users/Brian/Desktop/asd/.git/

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ |
```

Como puedes ver la consola nos muestra en rojo los archivos a los cuales no les está haciendo un control de cambios

Para que git les haga un seguimiento a estos archivos escribimos el comando



git add .

Para verificar su estado procedemos a poner otra vez git status y los archivos en rojo aparecen en verde.

```
MINGW64:/c/Users/Brian/Desktop/asd
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git add .

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git status
On branch master

No commits yet

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

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ |
```

Una vez se adicionan los archivos con el comando git add . estos archivos se añaden a una zona llamada Stage, que es una zona intermedia donde están los archivos listos para ser confirmados. Ahora para confirmar dichos archivos lo hacemos a través del comando:

git commit -m "Primer commit"

Como puedes ver se pone -m para indicar un mensaje, esto es recomendado para describir los cambios que se hicieron en esa versión.

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git commit -m "Primer commit"
[master (root-commit) a79d7ca] Primer commit
1 file changed, 9 insertions(+)
create mode 100644 index.html
```

Si hacemos otra vez un git Status, nos dirá que no se han realizado cambios, debido a que desde que se hizo el git commit el proyecto no ha sufrido ninguna modificación. Si hacemos una modificación y ponemos git status el archivo al que se le hizo la modificación aparecerá en rojo.



Hagamos la prueba y añadiremos una nueva línea a nuestro proyecto.

```
C: > Users > Brian > Desktop > asd > index.html > h
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Usando Git</title>
5      </head>
6      <body>
7          <h1>SoftFactory</h1>
8          <p>Esto es un ejemplo</p>
9      </body>
10 </html>
```

Como muestra en la siguiente imagen, la consola nos dice que el fichero index.html ha sido modificado

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Si queremos ver cuales fueron los cambios que se hicieron lo hacemos mediante el siguiente comando

git diff

Con este comando podremos ver en verde lo que se haya añadido al proyecto y en rojo lo que se haya quitado. En este caso solamente añadimos por lo que el resultado es el siguiente.

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git diff
diff --git a/index.html b/index.html
index 7b47802..b9197a1 100644
--- a/index.html
+++ b/index.html
@@ -5,5 +5,6 @@
     </head>
     <body>
       <h1>SoftFactory</h1>
+     <p>Esto es un ejemplo</p>
     </body>
   </html>
\ No newline at end of file
```



Si queremos volver a como teníamos nuestro último commit y eliminar los cambios que le hicimos a nuestro proyecto lo hacemos a través del comando

git checkout

No vamos a volver atrás procederemos a guardar los cambios. Para ello utilizamos otra vez git add . y git commit

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git add .

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html

Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git commit -m "Parrafo"
[master e360990] Parrafo
1 file changed, 1 insertion(+)
```

Como ya te has dado cuenta hemos realizado dos cambios en nuestro proyecto. Para ver estos cambios lo hacemos a través de

git log

Lo que nos mostrará el siguiente resultado

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git log
commit e36099056725682a82f6509af3a8982a36e03fc5 (HEAD -> master)
Author: Brian <brianpl990227@gmail.com>
Date:   Tue Aug 4 01:05:51 2020 +0100

    Parrafo

commit a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
Author: Brian <brianpl990227@gmail.com>
Date:   Tue Aug 4 00:45:04 2020 +0100

    Primer commit
```

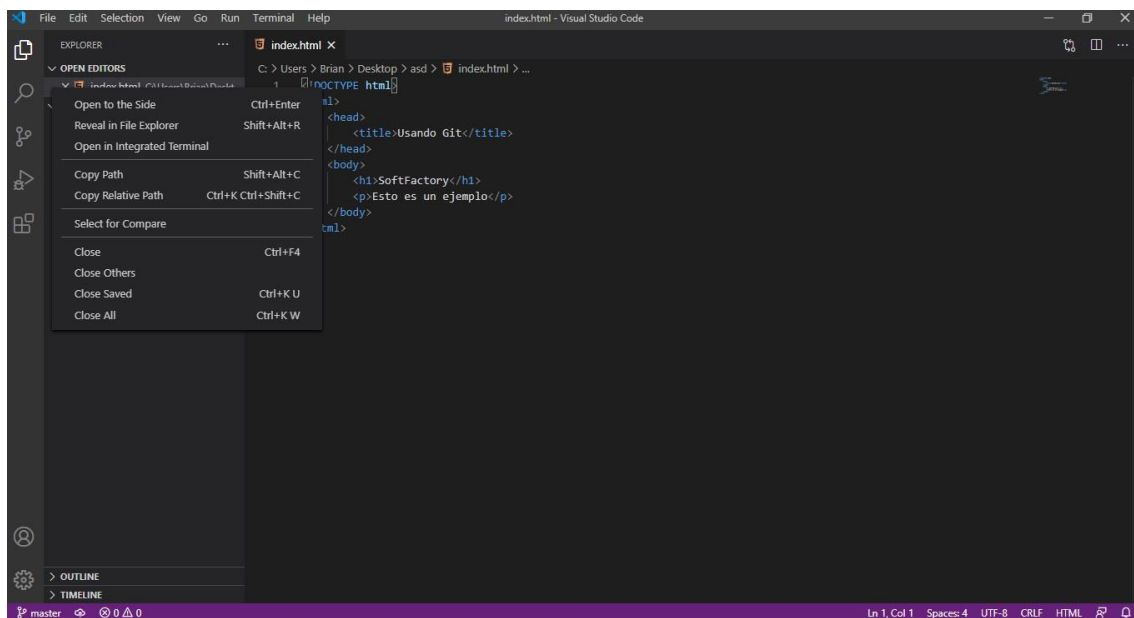


Ir a Hitos

Cabe mencionar ya que nos podemos equivocar al subir el mensaje del commit, que la forma de modificar este mensaje es a través del siguiente comando:

```
git commit --amend -m "Nuevo mensaje modificado"
```

Procedemos a poner un ejemplo. Vamos a eliminar todo lo que está dentro de nuestra etiqueta body en nuestro index.html y hagamos un commit con el mensaje "Este no" y luego otro con el mensaje "Texto modificado". Esta vez operaremos desde la consola del Visual Studio Code. Para ello damos click derecho en nuestra carpeta de proyecto y seleccionamos la opción abrir con Code. Una vez estemos dentro del editor de texto, damos click derecho en el explorer del proyecto y seleccionamos la opción Open in Integrated Terminal



Cuando demos click ahí podremos ver la siguiente consola donde escribiremos los comandos de git



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a file named 'index.html' under a folder named 'ASD'. The main editor area displays the content of 'index.html', which is a basic HTML document with a title 'Usando Git'. The terminal at the bottom shows the PowerShell prompt 'PS C:\Users\Brian\Desktop\asd>'.

```
index.html x
index.html > html > body
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Usando Git</title>
5   </head>
6   <body>
7
8   </body>
9 </html>
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL 1: powershell

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

PS C:\Users\Brian\Desktop\asd>

Como se muestra en la siguiente imagen adicionamos los cambios al Stage y tenemos en texto “Este no” en el último commit.

The screenshot shows the same Visual Studio Code interface as before, but the terminal now displays the output of several git commands. The HTML file content remains the same.

```
index.html x
index.html > html > body
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Usando Git</title>
5   </head>
6   <body>
7
8   </body>
9 </html>
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL 1: powershell

```
no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Brian\Desktop\asd> git add .
PS C:\Users\Brian\Desktop\asd> git commit -m "Este no"
[master 3bbf4a2] Este no
1 file changed, 1 insertion(+), 2 deletions(-)
PS C:\Users\Brian\Desktop\asd> git log
commit 3bbf4a269cd8274292b26fa5617a6d7f4eb3cae2
Author: Brian <brianp1990227@gmail.com>
Date: Tue Aug 4 23:55:25 2020 +0100

    Este no

commit e36099056725682a82f6509af3a8982a36e03fc5
Author: Brian <brianp1990227@gmail.com>
Date: Tue Aug 4 01:05:51 2020 +0100

    Parrafo

commit a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
```

Ahora para cambiar este texto por “Texto modificado” se procede a usar el comando de git

git commit --amend -m “Texto modificado”



```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL

PS C:\Users\Brian\Desktop\asd> git log
commit 114a6a88b1a2dad4fce5e83f74edcb2c19b4e672
Author: Brian <brianpl990227@gmail.com>
Date: Tue Aug 4 23:55:25 2020 +0100

    Texto modificado

commit e36099056725682a82f6509af3a8982a36e03fc5
Author: Brian <brianpl990227@gmail.com>
Date: Tue Aug 4 01:05:51 2020 +0100

    Parrafo

commit a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
Author: Brian <brianpl990227@gmail.com>
Date: Tue Aug 4 00:45:04 2020 +0100

    Primer commit
PS C:\Users\Brian\Desktop\asd> |
```

Como puedes ver hemos hecho ya varios commits a nuestro proyecto, como muestra en la imagen anterior cada commit tiene un id al lado. Si quisiéramos virar por ejemplo hacia nuestro primer commit, sería con el uso de ese id en el siguiente comando:

```
git reset --hard a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
```

Recordemos que todo el contenido de nuestro body fue eliminado, pues si hacemos uso del comando anterior el resultado es el siguiente

```
index.html x
index.html > html > body
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Usando Git</title>
5    </head>
6    <body>
7      <h1>SoftFactory</h1>
8    </body>
9  </html>

DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL

Author: Brian <brianpl990227@gmail.com>
Date: Tue Aug 4 01:05:51 2020 +0100

    Parrafo

commit a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
Author: Brian <brianpl990227@gmail.com>
Date: Tue Aug 4 00:45:04 2020 +0100

    Primer commit
PS C:\Users\Brian\Desktop\asd> git reset --hard a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
HEAD is now at a79d7ca Primer commit
```




Como puedes ver el proyecto volvió en el tiempo a como estaba inicialmente. Si hacemos uso del comando `git log` no se podrán ver los commits que se hicieron después del primer commit para ello utilizaremos el siguiente comando:

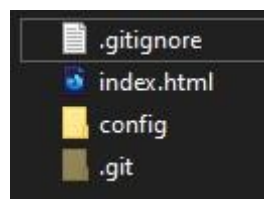
`git reflog`

Arrojando el siguiente resultado

```
PS C:\Users\Brian\Desktop\asd> git reflog
a79d7ca HEAD@{0}: reset: moving to a79d7ca08de4ec55b79c6b11b36ff9fb25beeb90
114a6a8 HEAD@{1}: commit (amend): Texto modificado
3bbf4a2 HEAD@{2}: commit: Este no
e360990 HEAD@{3}: commit: Parrafo
a79d7ca HEAD@{4}: commit (initial): Primer commit
PS C:\Users\Brian\Desktop\asd> █
```

Uso del gitignore

Muchas veces cuando creamos algún proyecto nuestro IDE genera archivos que no son necesarios para el desarrollo del proyecto. Ahora crearemos un `.txt` donde imaginaremos que están las configuraciones de ese IDE con el nombre de `Settings`, todo esto dentro de una carpeta llamada `config`. Para hacer ignorar esta carpeta tenemos que crear un archivo con la extensión `.gitignore`, es muy importante crear este archivo al mismo nivel de tu carpeta `.git` como se muestra en la siguiente imagen



Si hacemos un `git status` nos mostrará que git quiere llevar el control de la carpeta `config`, pero eso nosotros sabemos que no es necesario por lo que dentro de `.gitignore` escribimos la ruta de la carpeta `config` de la siguiente forma.

`/config/`

Si hacemos ahora un `git status`, no nos marcará la carpeta `config` ni nada de lo que esté en su interior

```
PS C:\Users\Brian\Desktop\asd> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Brian\Desktop\asd> █
```



No solo estás limitado al uso de rutas en el .gitignore, también puedes poner extensions y nombres de archivos como se muestra a continuación

Settings → nombre de archivo específico

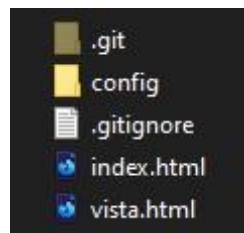
Settings* → Todo archivo que comience con Settings

.pepito → Todo archivo que tenga la extensión pepito

Ramas y Uniones

Una rama no es más que una división del desarrollo principal del software, con el fin de incorporar nuevos cambios que no estás seguro que formen parte de la rama principal o rama master. En este documento explicaremos el primer tipo de unión o merge conocido como Fast-Forward. Este tipo de merge al detectar que no hay ningún cambio sobre nuestra rama principal, puede hacer la unión de la rama principal con la secundaria.

Imaginemos que queremos añadir una nueva vista a nuestro proyecto, pero no estamos seguros de sí dejarla o no. Vamos a crear el archivo vista.html y nuestra carpeta del proyecto quedaría de la siguiente forma.



Si hacemos uso de lo aprendido anteriormente y hacemos un git status nos va a marcar el archivo vista.html en rojo, indicando que git quiere hacer control de él. Si no queremos modificar la rama principal con este nuevo archivo procedemos a utilizar el siguiente comando:

```
git branch nueva_vista
```

Con el que creamos una rama llamada nueva_vista. Entonces para ver estas ramas y saber en cual nos encontramos en este mismo instante usamos el comando

```
git log --oneline --decorate --all --graph
```

Lo que arrojará lo siguiente

```
PS C:\Users\Brian\Desktop\asd> git branch nueva_vista
PS C:\Users\Brian\Desktop\asd> git log --oneline --decorate --all --graph
* ecd07f3 (HEAD -> master, nueva_vista) GitIgnore
* a79d7ca Primer commit
```

Como puedes ver tenemos dos ramas en nuestro proyecto, la rama master y la rama nueva_vista. Esa salida en consola nos muestra también en que rama nos encontramos actualmente, en la rama master. Si queremos cambiar de rama lo hacemos mediante el comando:

```
git checkout nueva_vista
```




Una vez hayamos puesto el comando anterior y volvemos a hacer un git log –online –decorate –all –graph nos mostrará que estamos ya en la rama que creamos.

```
PS C:\Users\Brian\Desktop\asd> git checkout nueva_vista
Switched to branch 'nueva_vista'
PS C:\Users\Brian\Desktop\asd> git log --online --decorate --all --graph
* ecd07f3 (HEAD -> nueva_vista, master) GitIgnore
* a79d7ca Primer commit
```

Ya que estamos en la rama nueva_vista podemos hacer tranquilamente un commit hacia esta rama, sin que afecte a la rama master. Si hacemos nuevamente un git log –online –decorate –all –graph podemos ver el commit que se ha hecho solamente en esta rama.

```
PS C:\Users\Brian\Desktop\asd> git log --online --decorate --all --graph
* 5ecd9c5 (HEAD -> nueva_vista) Primer commit de la rama nueva vista
* ecd07f3 (master) GitIgnore
```

Como puedes ver nos dice que estamos en la rama nueva_vista y que tenemos un commit en esa rama, si te das cuenta también nos muestra los commits hechos en la rama master. Ahora hay que analizar que la rama que está controlando el archivo vista.html es la rama nueva_vista. Si hacemos un git checkout master, cambiaremos de rama y el archivo desaparecerá, pero si volvemos a la rama nueva_vista el archivo volverá a aparecer. Supongamos que te convenciste de que quieres añadir el archivo vista.html al proyecto, pues para eso hacemos la unión o merge de la rama nueva_vista con la rama master, para hacerlo es necesario posicionarnos en la rama master con el comando git checkout master y proceder a poner en consola:

```
git merge nueva_vista
```

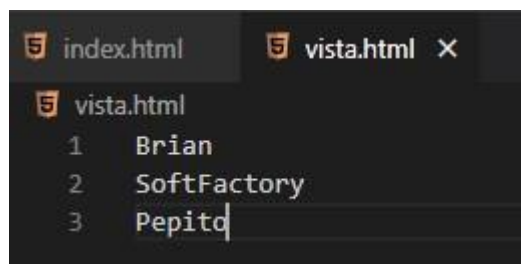
Por tanto, si ya no se desea tener la rama nueva_vista la podemos eliminar, siempre que estemos posicionados en la rama master con el siguiente comando

```
git branch -d nueva_vista
```

Es válido decir que si queremos crear una rama y automáticamente ir a ella en un solo comando lo podemos hacer de la siguiente forma

```
git checkout -b nueva_rama
```

Hay un caso en particular donde se trabaja en diferentes ramas pero con el mismo archivo. Esto genera problemas a la hora de hacer un git merge, estos problemas hay que solucionarlos manualmente. En nuestro archivo vista.html vamos a proceder a ponerle algunas palabras para demostrar de forma práctica lo antes planteado, recordemos que estamos posicionados en la rama master y eliminamos nueva_vista. Escribimos algo en el archivo vista.html



```
index.html  vista.html X
vista.html
1  Brian
2  SoftFactory
3  Pepito
```

Hacemos un commit y todo listo por la rama master. Ahora creamos una nueva rama llamada bum y haremos algunos cambios en el archivo vista.html



```
index.html  vista.html X
vista.html
1  Brian
2  SoftFactory
3  Fulano
4  Mengano
5  Pepito
```

Añadimos 2 líneas y procedemos a hacer un commit en la rama bum. Si cambiamos a la rama master, las 2 líneas añadidas se borrarán ya que cambiamos de rama.

```
index.html  vista.html X
vista.html
1  Brian
2  SoftFactory
3  Pepito
```

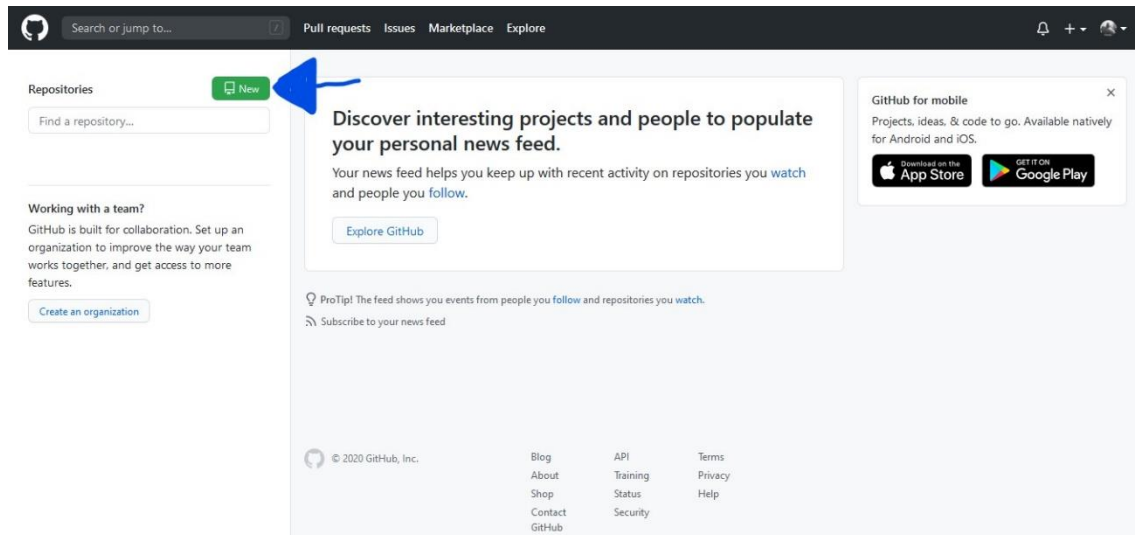
Ahora vamos a hacer el merge de la rama master con la rama bum y veamos que pasa.

```
vista.html > ?
1  Brian
2  SoftFactory
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3  <<<<<<< HEAD (Current Change)
4  fulano
5  mengano
6  =====
7  >>>>>>> bum (Incoming Change)
8  Pepito
```

Como muestra la imagen anterior se puede ver un conflicto. Para arreglar esto procedemos a presionar en el botón Accept Current Change para hacer la mezcla de los 2 códigos y solucionar el conflicto, si no la deseas hacer presionas en Accept Incoming Changes.

GitHub


GitHub es una red social la cual se basa en la tecnología de Git. Aquí podemos subir nuestro código en repositorios públicos y privados, así como clonarlo o actualizarlo con los cambios que han realizado otros desarrolladores. Esto permite un trabajo en equipo fluido y controlado. Una vez nos hayamos registrado en GitHub nos aparecerá la siguiente vista.



Donde tocaremos en el botón que indica en la imagen anterior para crear un nuevo repositorio donde subiremos el código de nuestro proyecto. Una vez toquemos en el botón New nos parecerá un formulario donde debemos de poner el nombre de nuestro repositorio.

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 [fantastic-rotary-phone?](#)

Description (optional)

Esto es un ejemplo de como usar GitHub

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

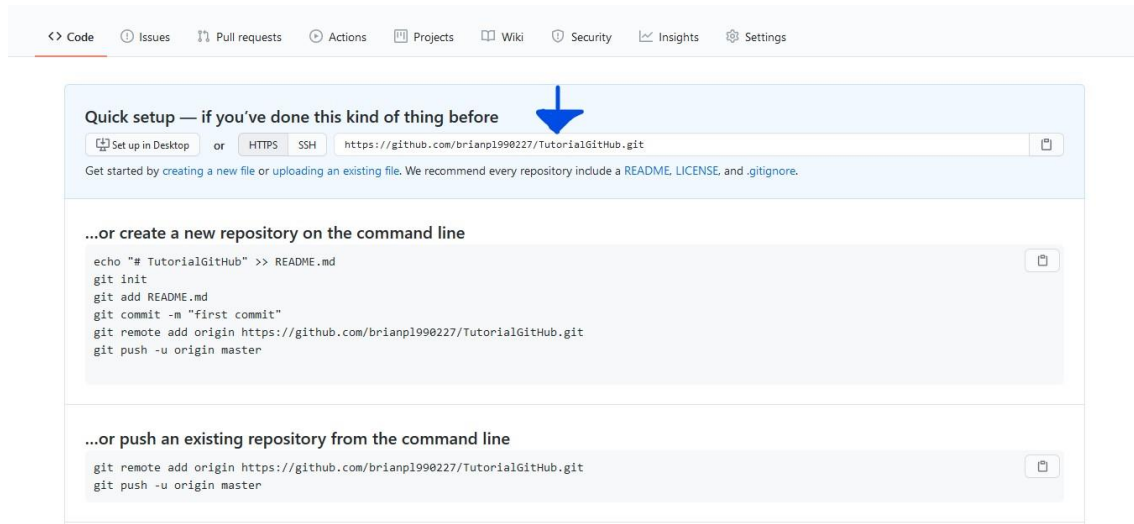
Skip this step if you're importing an existing repository.

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

Add .gitignore: None Add a license: None ⓘ

Create repository

Introducimos el nombre del repositorio, una descripción que es opcional, seleccionamos la opción de público o privado y con esa información ya podemos proceder a crear nuestro repositorio. Una vez creado nuestro repositorio GitHub nos da su dirección y como configurarlo. Así como se muestra a continuación.



En nuestro caso como ya tenemos un proyecto donde ya Git está haciendo un control de versiones y estamos al día con los cambios solo tenemos que poner los últimos dos comandos.

`git remote add origin https://github.com/brianpl990227/TutorialGitHub.git`

Para comprobar que nuestro remoto se hizo correctamente ponemos el comando

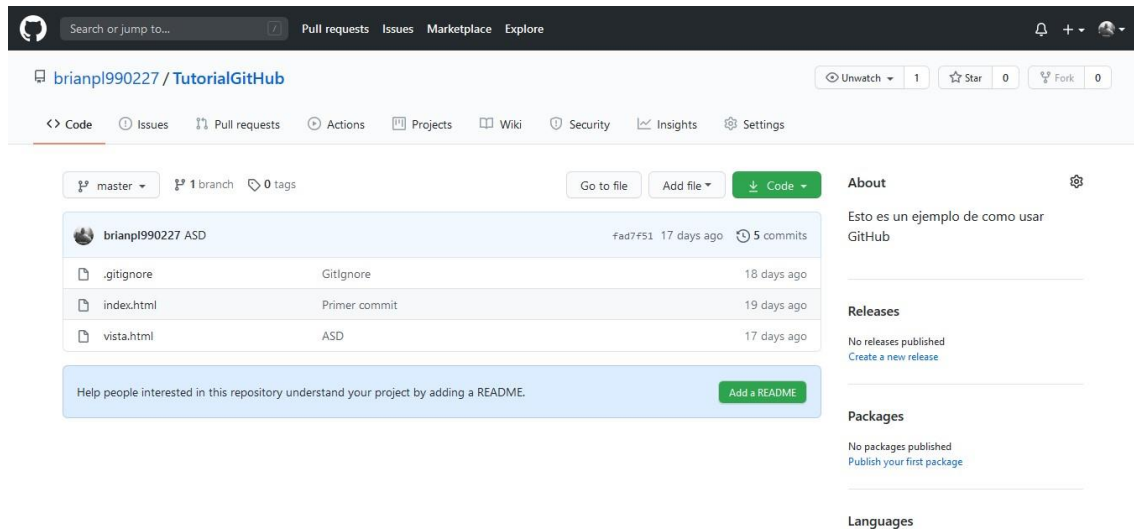
`git remote -v` y nos aparecerá lo siguiente

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master|MERGING)
$ git remote --v
origin https://github.com/brianpl990227/TutorialGitHub.git (fetch)
origin https://github.com/brianpl990227/TutorialGitHub.git (push)
```

No necesariamente tiene que llamarse origin podemos ponerle Pepito si queremos. Con ese comando conectamos nuestro Git a nuestro repositorio de GitHub. Pero eso no quiere decir que ya nuestro código está en GitHub, para ello necesitamos hacer un push con el siguiente comando

`git push origin master`

Como puedes ver en el comando anterior hicimos referencia al remoto que creamos llamado origin y que el código que tenemos se va a subir a la rama master de nuestro repositorio. Cuando hagamos eso en la consola o en el IDE nos va a pedir las credenciales de Github, donde ponemos nuestro nombre de usuario y contraseña de Github, si lo haces por la consola y a la hora de poner la contraseña no aparece nada esto es un comportamiento normal de las consolas por lo que sigue escribiendo tu contraseña y presiona enter cuando termines. Una vez realizado esto se hace el push y si refrescamos nuestro navegador se puede ver que nuestro código ya está en GitHub.



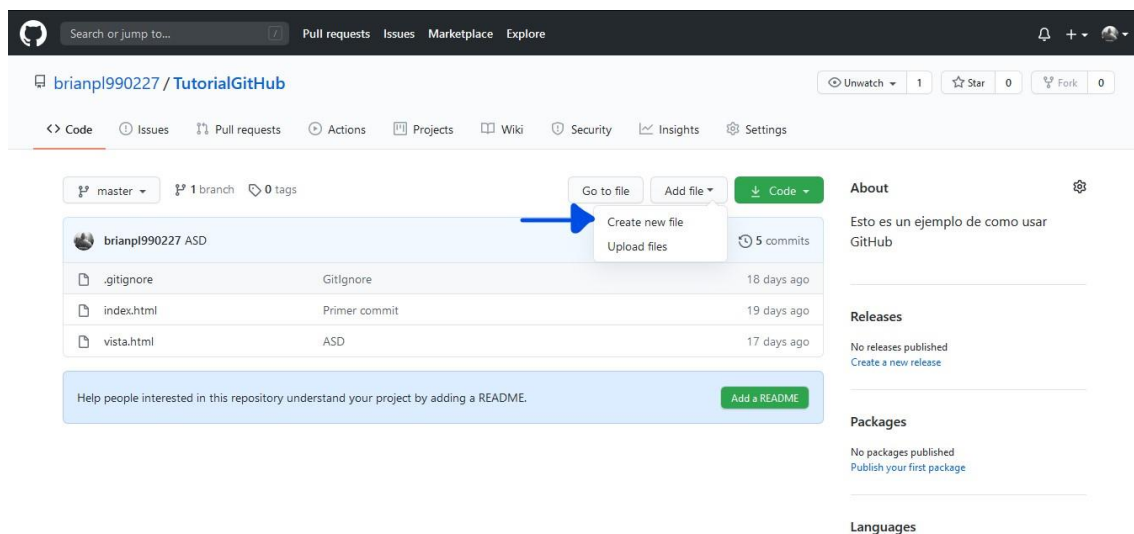
Como puedes ver en la imagen anterior hizo push de todo excepto lo que está en el .gitignore. Supongamos que te mueves hacia otra máquina o hayas borrado por accidente el proyecto. Puedes hacer simplemente uso del siguiente comando

git clone <https://github.com/brianpl990227/TutorialGitHub.git>

Si trabajamos con más personas en nuestro proyecto es posible que esas personas suban nuevas líneas o archivos de código a nuestro proyecto. Si hacemos un git status en nuestro proyecto local y no hemos cambiado nada nos va a decir que está al día, pero si trabajamos con un equipo existe de enorme posibilidad de que nuestro proyecto no esté al día.

Para ello hay dos formas de traer los cambios, haciendo un fetch y luego un merge o mediante el comando pull que es una forma más práctica de hacer esto donde los cambios se agregan automáticamente a la rama master de nuestro proyecto.

Vamos a simular el caso anterior creando un archivo directamente en GitHub llamado contact.html para ello tocamos en el botón AddFile



Una vez hayamos dado click en Create new file veremos la siguiente vista donde colocaremos el nombre del archivo y su contenido.



The screenshot shows the GitHub web interface for the repository 'brianpl990227 / TutorialGitHub'. The 'Commit new file' dialog is open, showing the file 'contact.html' with the content 'Esto es una prueba!'. The dialog has a 'Commit new file' button and a 'Cancel' button. The commit message field is empty, and the description field contains 'Aquí va una descripción'. The 'Commit directly to the master branch' option is selected.

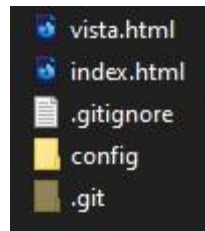
Ponemos también el nombre del commit y una descripción. Como puedes ver nos da la opción de añadir este archivo directamente en la rama master o en otra rama. En nuestro caso seleccionaremos hacerlo directamente en la rama master y tocamos en Commit new File.

The screenshot shows the GitHub web interface for the repository 'brianpl990227 / TutorialGitHub'. The commit history is displayed, showing the commit 'Hacer pull' by 'brianpl990227' with the SHA '115a7bb' and the message '115a7bb in 5 hours 6 commits'. The commit history table is as follows:

File	Commit Message	Time
.gitignore	Gitignore	18 days ago
contact.html	Hacer pull	now
index.html	Primer commit	19 days ago
vista.html	ASD	17 days ago

Below the table, there is a message: 'Help people interested in this repository understand your project by adding a README.' with a button 'Add a README'.

Tenemos el archivo contact.html supuestamente subido por otro desarrollador por lo que nuestro proyecto local está desactualizado.

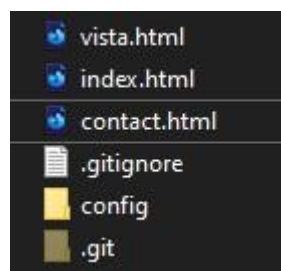


Como puedes ver en la imagen anterior no existe el archivo contact.html por lo que procedemos siempre que vayamos a trabajar en nuestro proyecto usar el siguiente comando

`git pull origin master`

Esto traerá los últimos cambios del repositorio de GitHub a nuestro proyecto local.

```
Brian@DESKTOP-V9UAJJK MINGW64 ~/Desktop/asd (master)
$ git pull origin master
From https://github.com/brianpl990227/TutorialGitHub
 * branch                master      -> FETCH_HEAD
hint: Waiting for your editor to close the file...
[main 2020-08-22T17:47:54.727Z] update#setState idle
Merge made by the 'recursive' strategy.
 contact.html | 1 +
1 file changed, 1 insertion(+)
create mode 100644 contact.html
```



Conclusiones

Podemos ver que con el uso de git se puede realizar un proyecto de software de una manera controlada, escalable y sencilla. Facilitando el trabajo en equipo de los desarrolladores y la revisión de los distintos hitos de nuestro proyecto en el tiempo.



SoftFactory

Desarrollo de software y
marketing digital & publicidad



softfactoryreads@gmail.com



@SoftFactory3



@softfactoryopcion



@softfactorydev