



# Practica:

## GuruDay - Git and GitHub

## **Contenido:**

1. *Objetivo.*
2. *¿Cómo instalar Git?*
3. *Repositorio local Git.*
  - a. *Crear repositorio.*
  - b. *Agregar archivos.*
4. *Ramificar y Fusionar.*
  - a. *Crear Ramas.*
  - b. *Moverse entre ramas.*
  - c. *Unir ramas.*
5. *Conectar con Github.*
  - a. *Crear conexión.*
  - b. *Subir al servidor.*
  - c. *Actualizar.*
  - d. *Traer del servidor.*

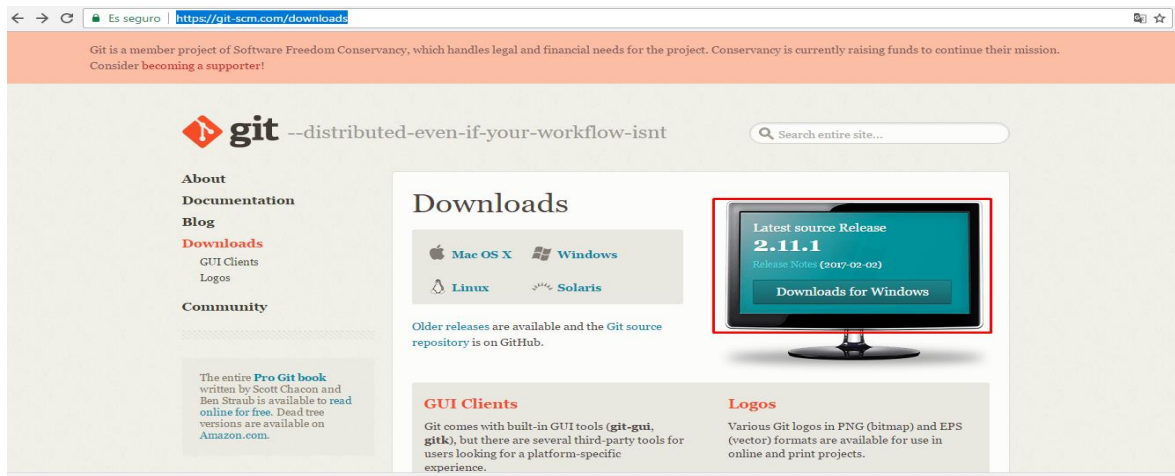
## **Objetivo.**

El objetivo de este manual es que el alumno ponga en práctica la teoría explicada en la presentación anexa a este documento. Al realizar las actividades de este manual el alumno será capaz de crear repositorios locales, hacer commits a los archivos de un proyecto, manejar cada una de las versiones que genere, crear, manipular y eliminar ramas, usar la función Merge de git para unir las ramas de un repositorio y aplicar todas las funciones anteriores en un repositorio alojado en un servidor.

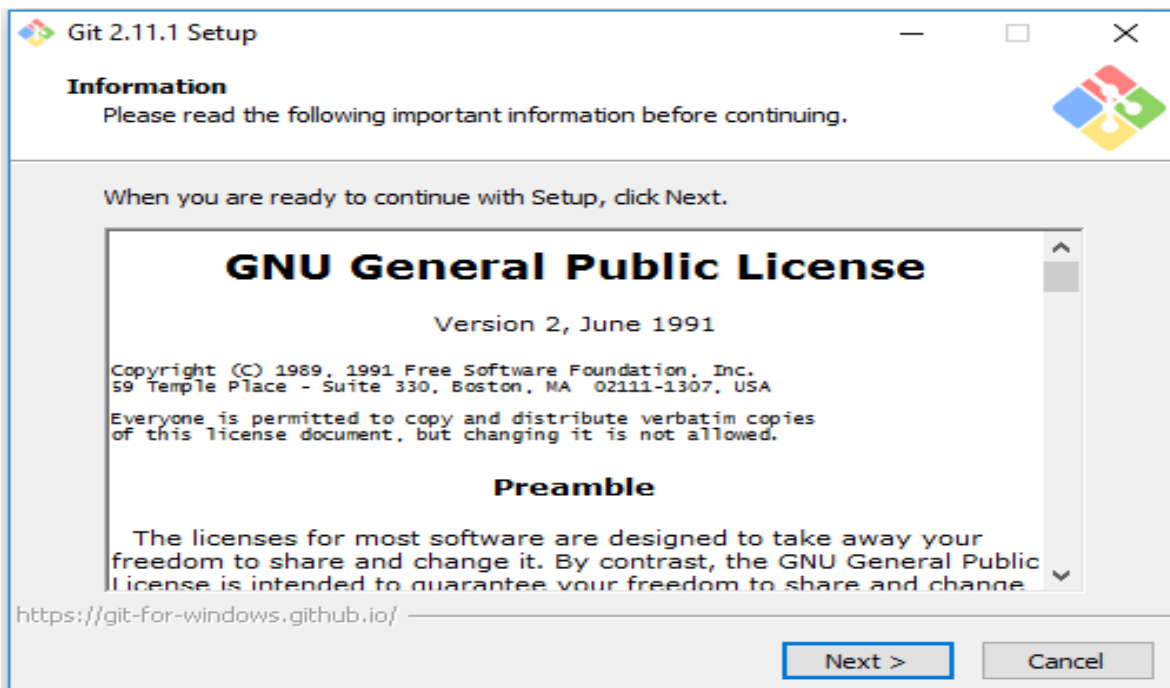
## ¿Cómo instalar git?

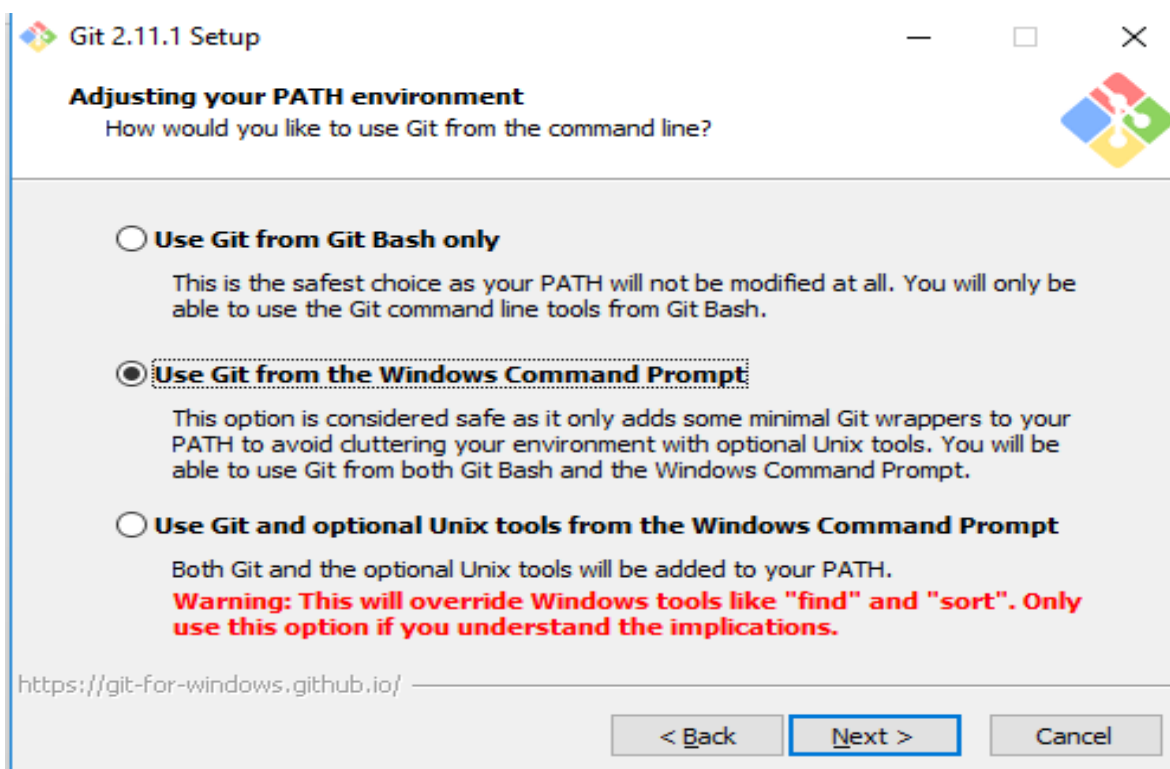
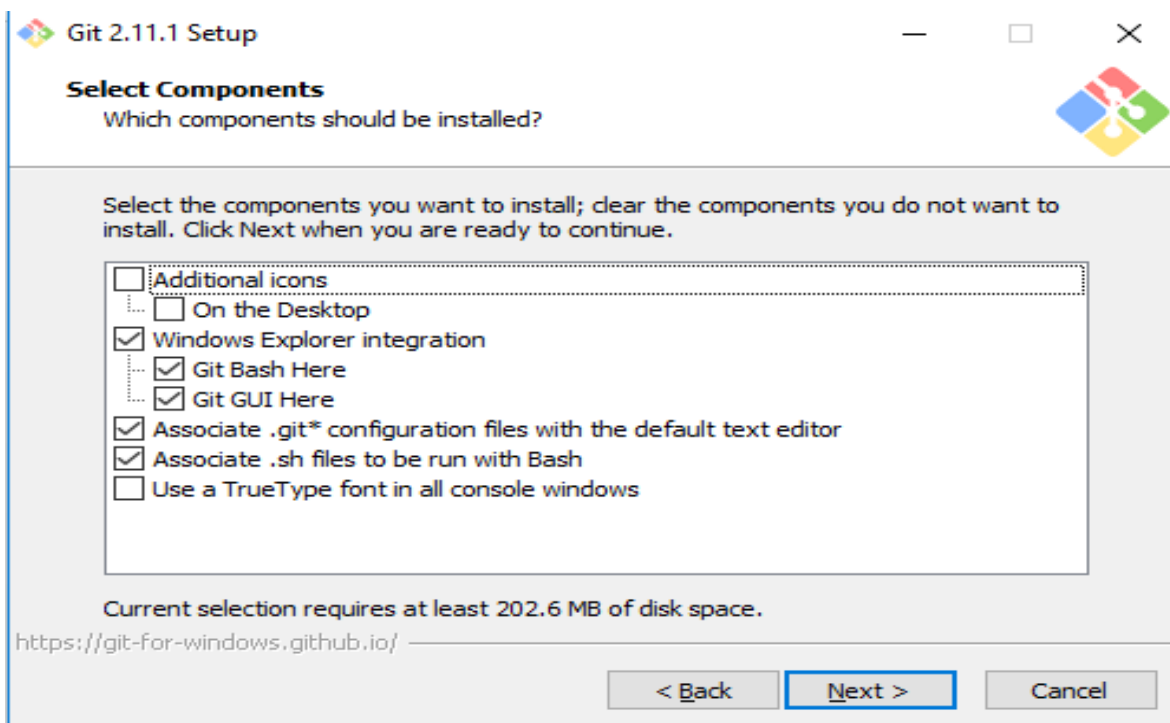
<https://git-scm.com/downloads>

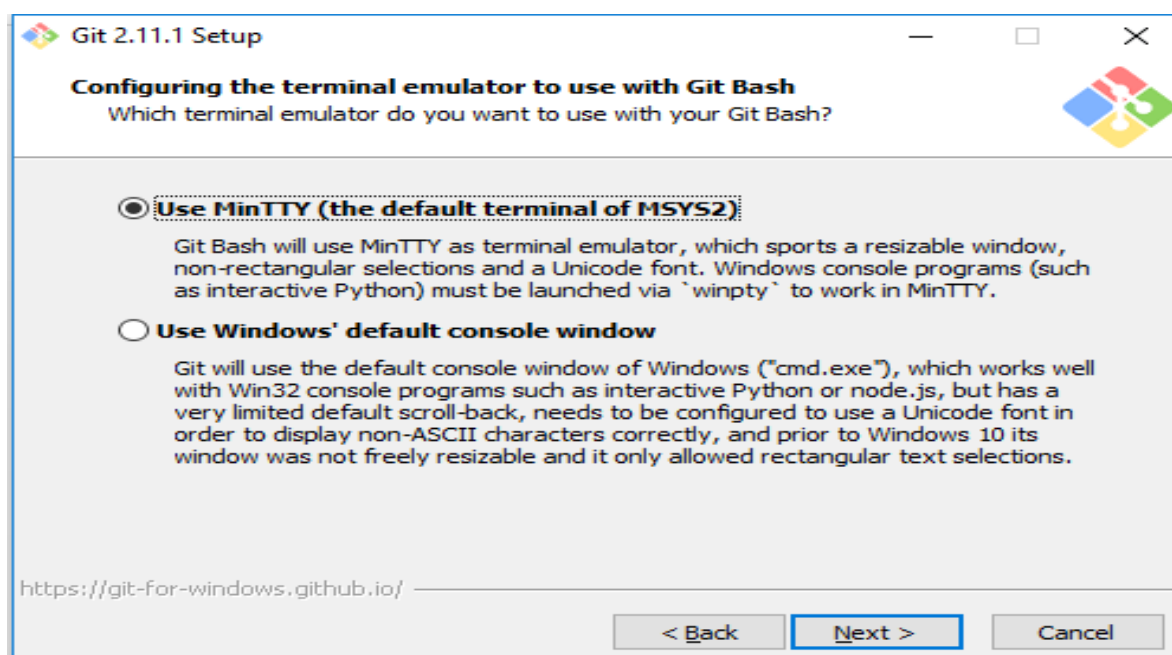
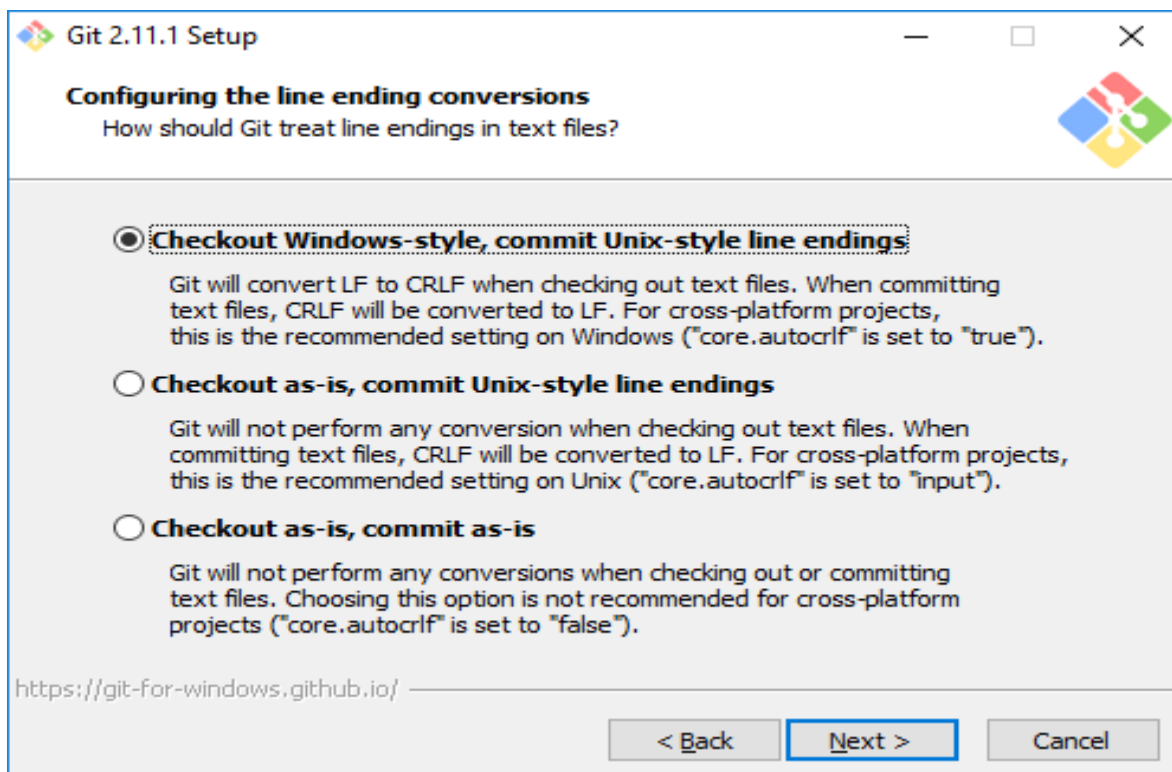
- Descargamos de la página oficial de Git, la versión para el S.O. que se requiera.

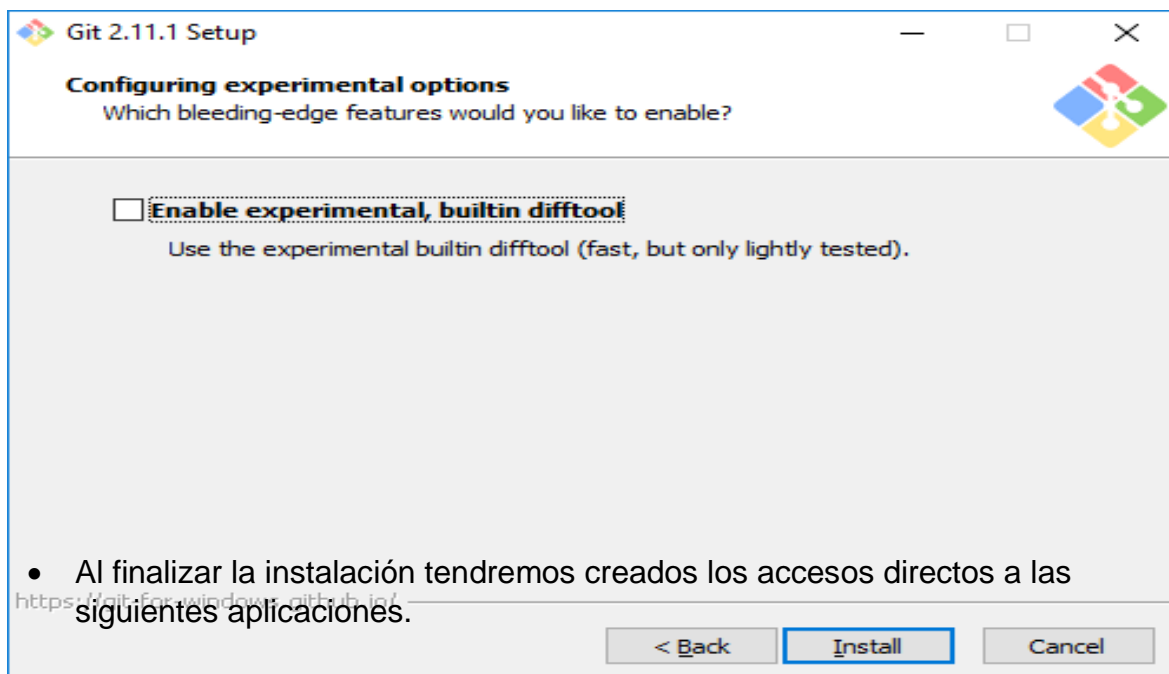
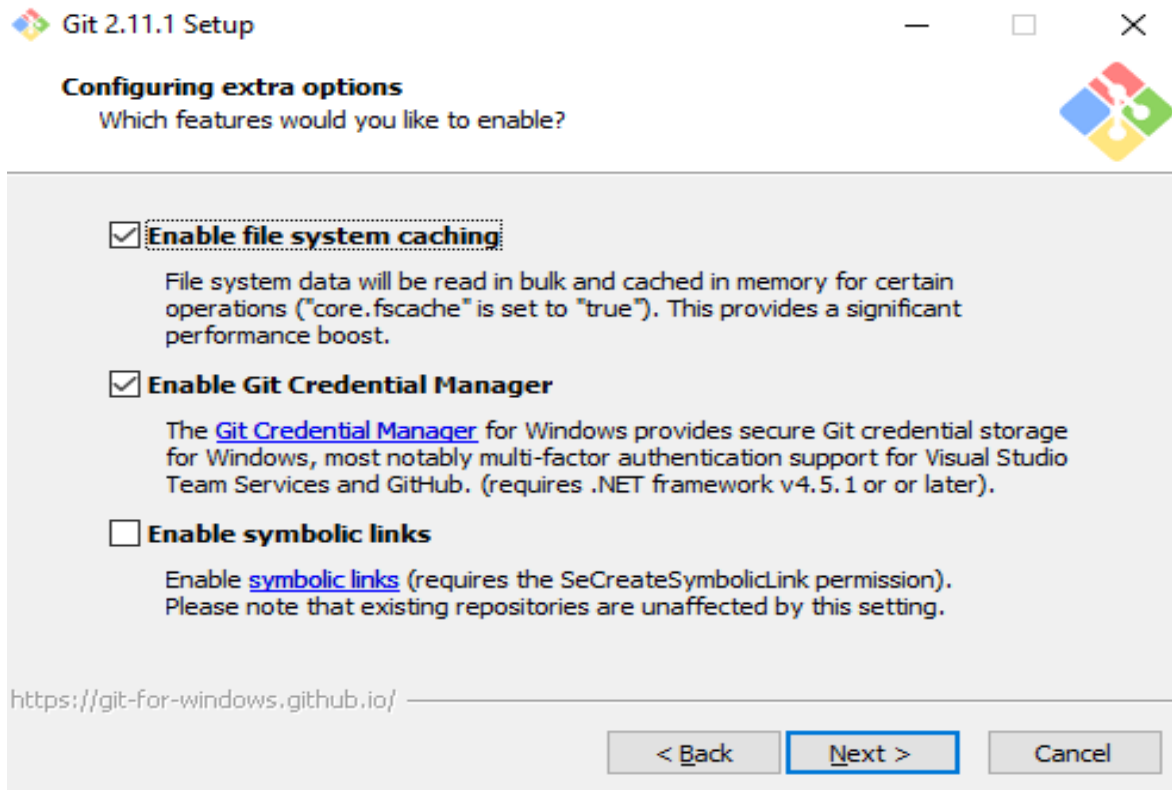


- Utilizar la configuración que se muestra por default.











- ❖ **Git Bash:** Es la consola de comandos de Git.
  - ❖ **Git CMD:** Consola del sistema mezclada con Git.
  - ❖ **Git GUI:** Es la interfaz gráfica de Git.
- Una forma de verificar si Git se instaló correctamente es con el comando: **“git version”** que muestra la versión de Git que tenemos instalada

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

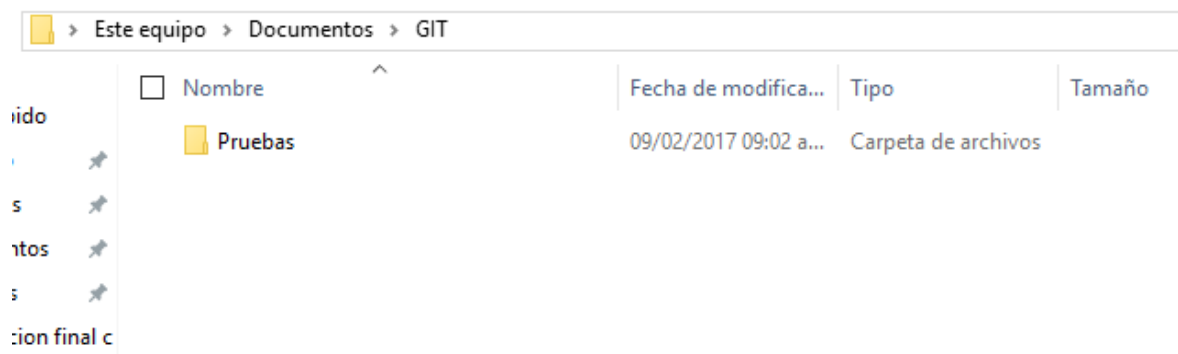
C:\Users\TecGurus-Workshop>git version
git version 2.11.1.windows.1

C:\Users\TecGurus-Workshop>
```

## Repositorio local Git.

### Crear repositorio.

- Se crea la carpeta donde se guardaran los repositorios.



- Abrimos **Git Bash**



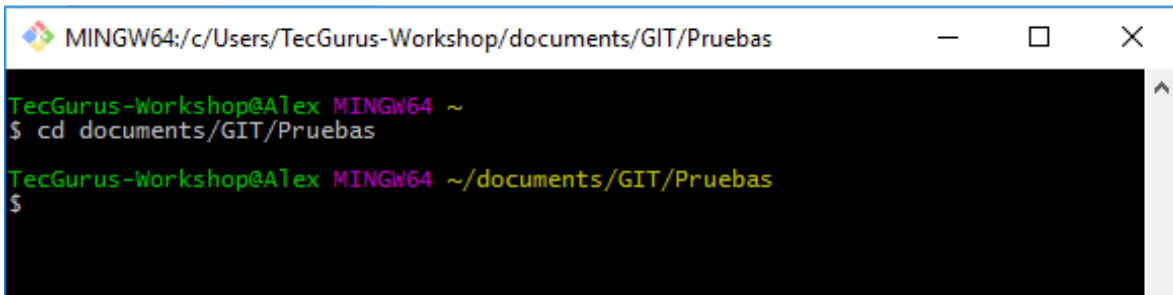
- NOTA:** Antes de iniciar con un **Commit** se recomienda agregar tu nombre de usuario y tu email para futuras referencias.

```
$ git config --global user.name "TecGurus"
```

```
$ git config --global user.email "git@tecgurus.net"
```

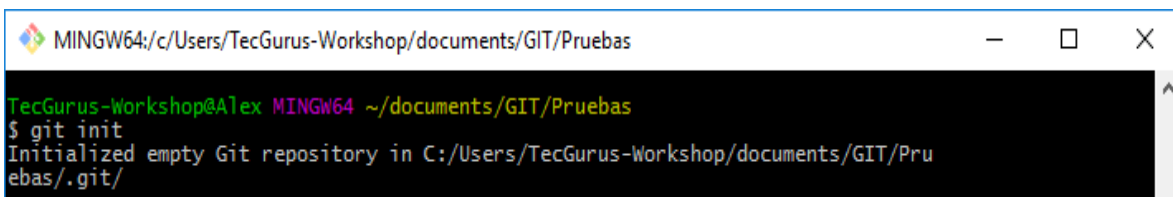


- Nos posicionamos dentro de la carpeta donde guardarás tu proyecto.



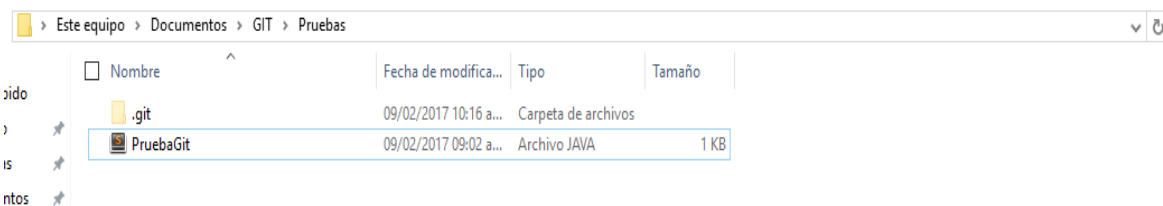
```
MINGW64:/c:/Users/TecGurus-Workshop/documents/GIT/Pruebas
TecGurus-Workshop@Alex MINGW64 ~
$ cd documents/GIT/Pruebas
TecGurus-Workshop@Alex MINGW64 ~/documents/GIT/Pruebas
$
```

- Usamos el comando **git init** para crear el repositorio.

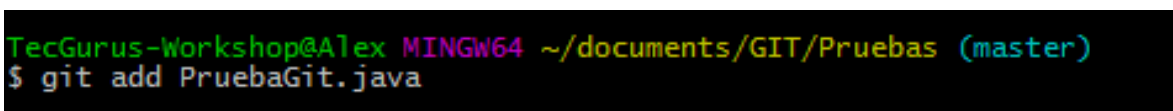


```
MINGW64:/c:/Users/TecGurus-Workshop/documents/GIT/Pruebas
TecGurus-Workshop@Alex MINGW64 ~/documents/GIT/Pruebas
$ git init
Initialized empty Git repository in C:/Users/TecGurus-Workshop/documents/GIT/Pruebas/.git/
```

- Se creará una carpeta **.git** en modo oculto, esta carpeta solo debe ser modificada con comandos de Git directo de la consola.



- Una vez creado el repositorio se mandan los archivos a la zona de índice con el comando "**git add**"



```
TecGurus-Workshop@Alex MINGW64 ~/documents/GIT/Pruebas (master)
$ git add PruebaGit.java
```



- Se consulta el histórico de commits con el comando "**git log**"

```
TecGurus-Workshop@Alex MINGW64 ~/documents/GIT/Pruebas (master)
$ git log
commit 981e77bb67fb609e2221d8d8eb9700dd24d4daf
Author: TecGurus <git@tecgurus.net>
Date: Thu Feb 9 11:59:58 2017 -0600

Este es nuestro siguiente commit

commit f5d530c50590375e23173d89d2600ca95e1ad628
Author: TecGurus <git@tecgurus.net>
Date: Thu Feb 9 11:56:58 2017 -0600

Este es un commit de prueba

commit 9b8431dcfc78b55eabe9297a79ada2cc72a05c0d
Author: TecGurus <git@tecgurus.net>
Date: Thu Feb 9 11:17:08 2017 -0600

Primer Commit
```

- Para un log simplificado usamos "**git log --oneline**"

```
TecGurus-Workshop@Alex MINGW64 ~/documents/GIT/Pruebas (master)
$ git log --oneline
981e77b Este es nuestro siguiente commit
f5d530c Este es un commit de prueba
9b8431d Primer Commit
```

Hasta el momento sabemos cómo crear repositorios locales, agregarlos a la zona de índice y generar los commits que darán seguimiento al desarrollo de nuestro sistema, ahora necesitamos saber cómo regresar a una versión anterior de nuestro desarrollo, esto lo podemos lograr con el comando **checkout**.

- Para tener un seguimiento más detallado de nuestros commits se usa el comando: `git log --oneline --decorate`

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git log --oneline --decorate
edca8d2 (HEAD -> master, origin/master, origin/HEAD) Prueba 4
c402a73 Este es el commit con error (falta -m)
53a8730 Segundo Commit de Prueba
47aae70 Primer Commit
```

- El código en amarillo es un identificador único para cada commit, lo usaremos con el comando checkout para regresar a una versión anterior.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git checkout 53a8730
```

- La etiqueta HEAD indica cual es la versión sobre la que estamos trabajando.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java ((53a8730...))
$ git log --oneline --decorate
53a8730 (HEAD) Segundo Commit de Prueba
47aae70 Primer Commit
```

- Para regresar a la última actualización basta con hacer un **checkout** a la rama principal.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java ((53a8730...))
$ git checkout master
Your branch is up-to-date with 'origin/master'.
Previous HEAD position was 53a8730... Segundo Commit de Prueba
Switched to branch 'master'

Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git log --oneline --decorate
edca8d2 (HEAD -> master, origin/master, origin/HEAD) Prueba 4
c402a73 Este es el commit con error (falta -m)
53a8730 Segundo Commit de Prueba
47aae70 Primer Commit
```

- Para identificar cada commit creados podemos usar el comando **tag**, que nos genera una etiqueta con la cual podemos localizar un commit en específico sin mayor problema.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git tag v1.0 47aae70

Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git log --oneline --decorate
edca8d2 (HEAD -> master, origin/master, origin/HEAD) Prueba 4
c402a73 Este es el commit con error (falta -m)
53a8730 Segundo Commit de Prueba
47aae70 (tag: v1.0) Primer Commit
```

- Con la etiqueta podemos hacer un checkout de manera sencilla.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git checkout v1.0
```

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java ((v1.0))
$ git log --oneline --decorate
47aae70 (HEAD, tag: v1.0) Primer Commit
```

## Ramificar y Fusionar

### Crear ramas

- Para crear una rama usamos el comando branch.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git branch rama1
```

- Podemos ver las ramas creadas al momento simplemente utilizando el comando branch.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git branch
* master
  rama1
  rama2
  rama3
```

- Para movernos entre las ramas creadas usaremos el comando checkout y el nombre de la rama a la que queremos entrar.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git checkout rama2
Switched to branch 'rama2'

Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (rama2)
$
```

- Para cambiar el nombre de las ramas seguimos la siguiente sintaxis:  
*\$ Git branch -m nombre-rama nombre-rama-nueva*

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (rama2)
$ git branch -m rama1 bug

Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (rama2)
$ git branch
bug
master
* rama2
rama3

Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (rama2)
$ |
```

## Unir ramas

- Para unir las ramas que se han creado, primero, tenemos que posicionarnos en la rama a la que se pretende integrar, en este caso a la rama “master”.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (rama2)
$ git checkout master
Your branch is up-to-date with 'origin/master'.
Switched to branch 'master'
```

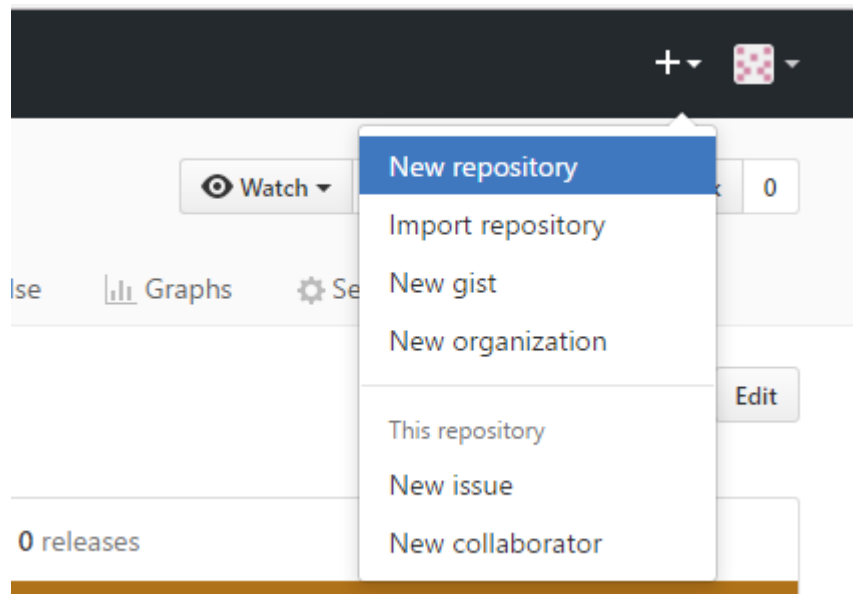
- Después se integra la rama adicional con la rama “master” con el comando Merge.

```
Joaquin@PerlaNegra MINGW64 ~/documents/GIT/java (master)
$ git merge rama2
Updating edca8d2..54a99ae
Fast-forward
 Prueba.java | 1 +
 1 file changed, 1 insertion(+)
```

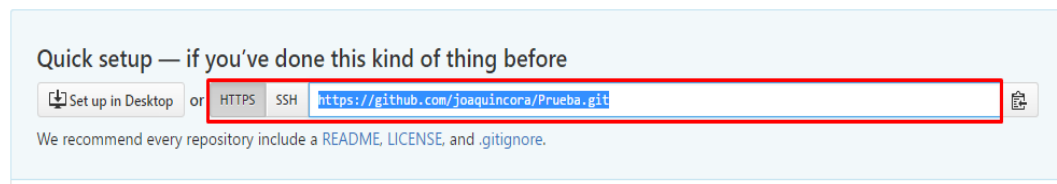
## Conectar con Github.

### Crear conexión

- Para subir el repositorio a un servidor (en este caso Github) primero tendremos que crear una cuenta.
- Una vez hecha la cuenta se agrega un nuevo repositorio con tu usuario de Github.



- Ya creado el nuevo repositorio en tu cuenta copiamos la URL del repositorio.



- después damos de alta el repositorio en la consola de git con el siguiente comando:

*Git remote add prueba <URL del repositorio>*

```
Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$ git remote add prueba https://github.com/joaquincora/Prueba.git
```

- Para ver todas las conexiones a repositorios remotos que tenemos utilizamos el comando:

*Git remote -v*

```
Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$ git remote -v
prueba https://github.com/joaquincora/Prueba.git (fetch)
prueba https://github.com/joaquincora/Prueba.git (push)
```

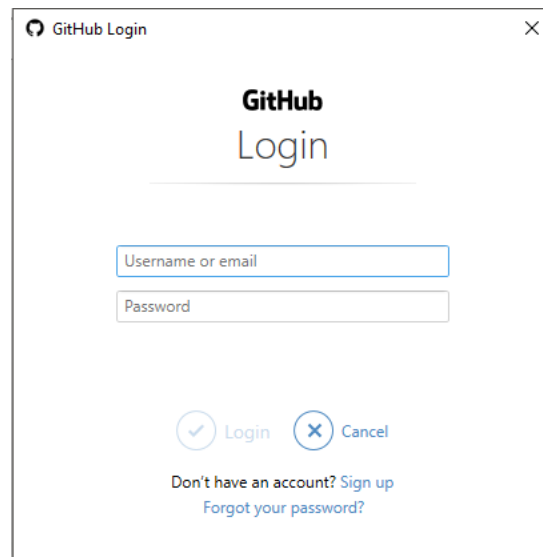
## Subir al servidor

- Para subir los archivos al servidor usaremos el comando push como se muestra en la imagen.

```
Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$ git push prueba master
Logon failed, use ctrl+c to cancel basic credential prompt.
Username for 'https://github.com/': joaquincora
Counting objects: 3, done.
Writing objects: 100% (3/3), 214 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/joaquincora/Prueba.git
 * [new branch]      master -> master
```



- Después de ingresar tu usuario y contraseña se habrá enviado el proyecto al servidor.



## Actualizar

- Cuando en el servidor existe una versión del código que no tenemos de manera local tenemos que descargar esa actualización, lo cual conseguimos con el comando pull.

```
Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$ git pull prueba master
From https://github.com/joaquincora/Prueba
* branch          master       -> FETCH_HEAD
Already up-to-date.

Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$
```

## Traer del servidor

- En caso de necesitar descargar todo el repositorio completo usaremos el comando clone, este comando trae una copia completa del repositorio almacenado en el servidor al repositorio local. Usaremos el siguiente comando:

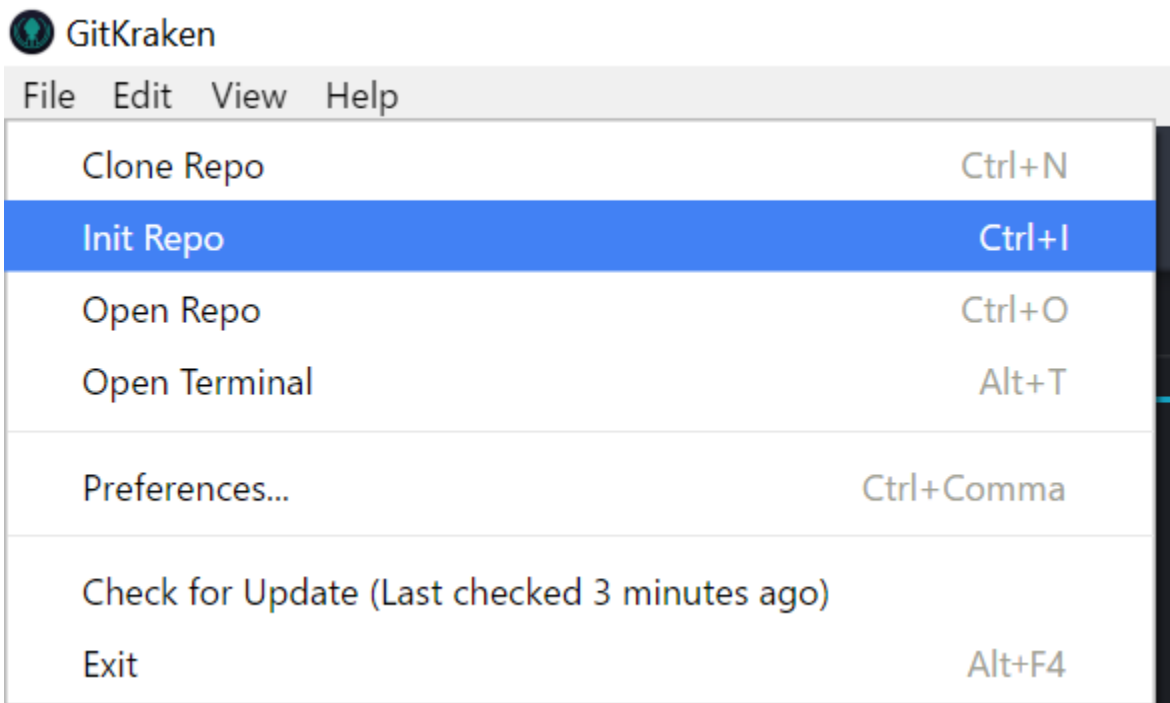
*Git clone <URL del repositorio*

```
Joaquin@PerlaNegra MINGW64 ~/documents/git/RepGit (master)
$ git clone https://github.com/joaquincora/java.git
```

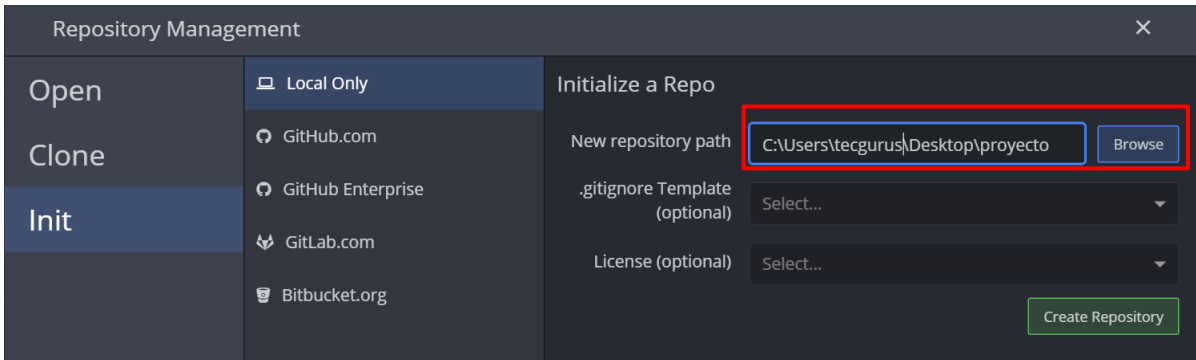
## Git Kraken

Para crear un repositorio con Git Kraken seguiremos la siguiente ruta:

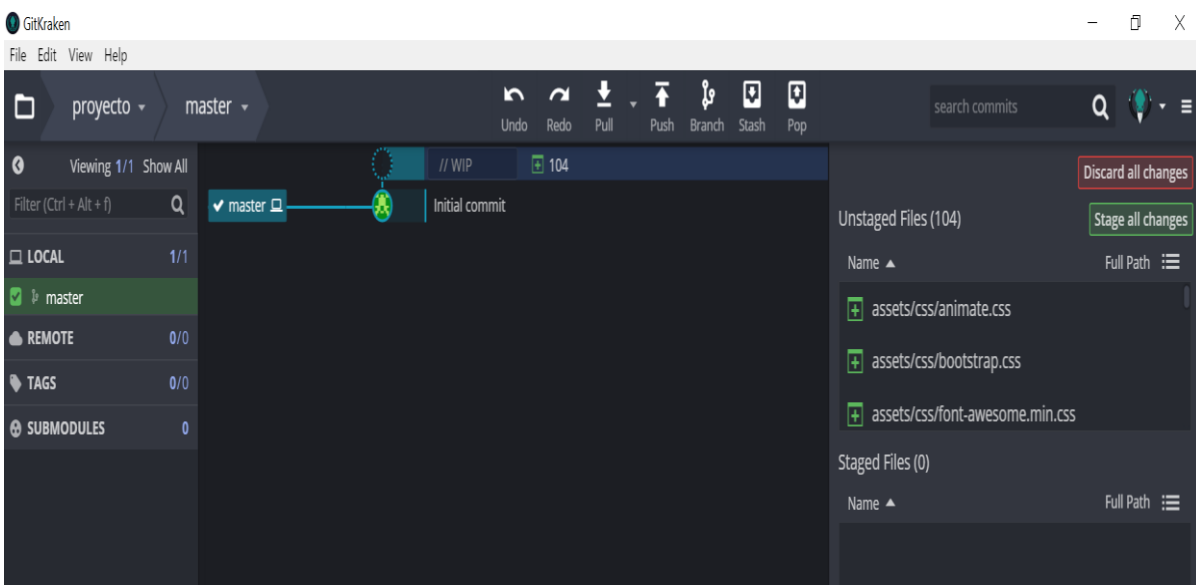
**File>>Init Repo**



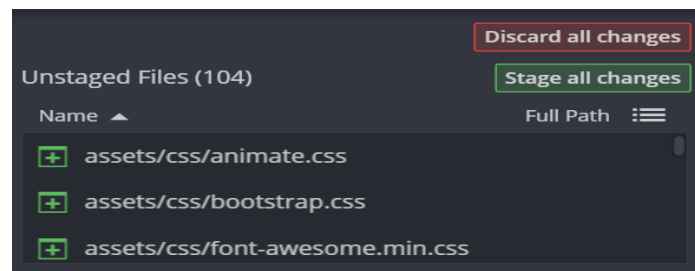
Haremos clic en Init y luego en “Local Only”, en la parte señalada de color rojo buscaremos la carpeta de nuestro proyecto para sí poder crear el repositorio.



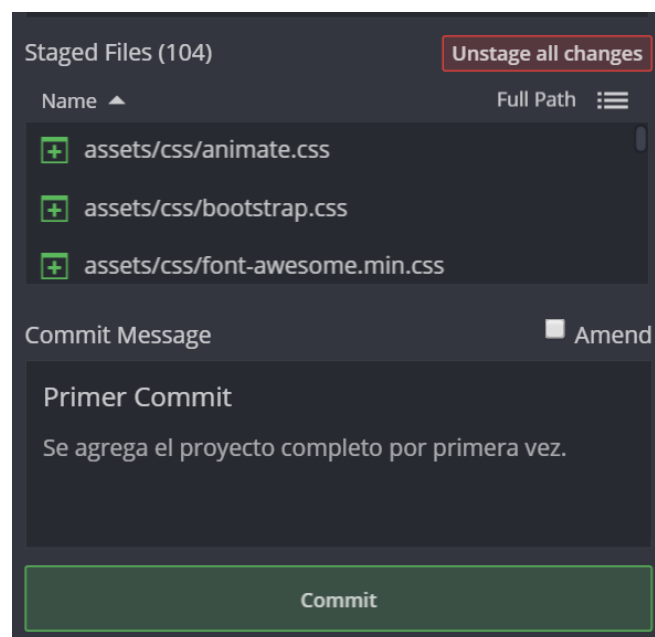
Se mostrará la siguiente pantalla, esto indica que el repositorio fue creado de manera correcta:



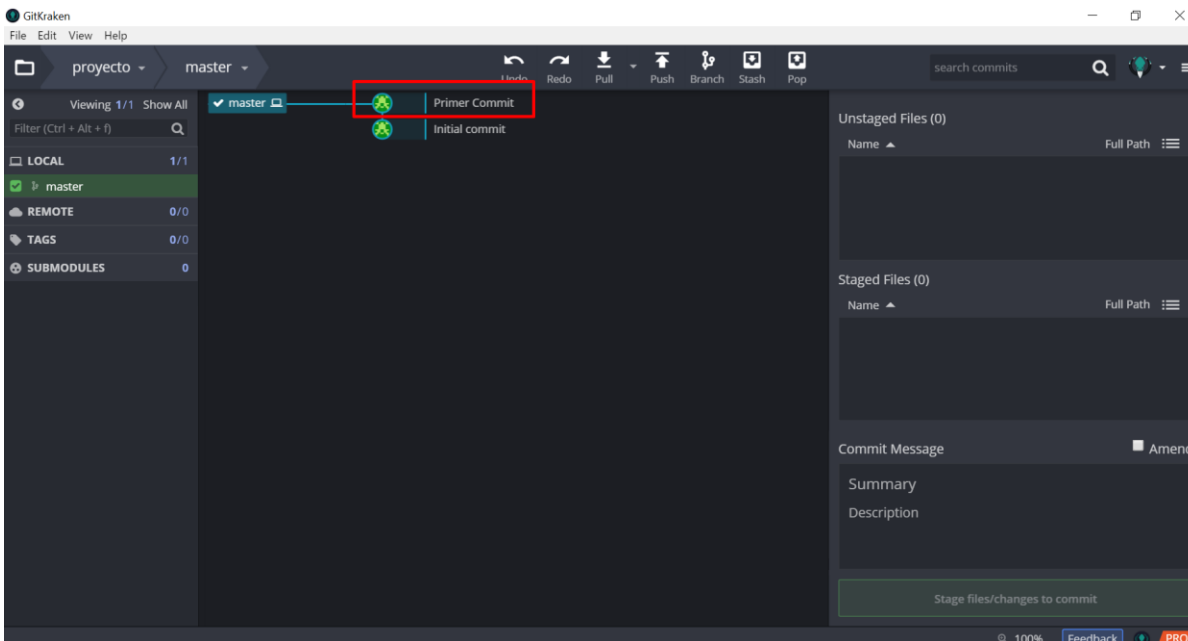
Del lado derecho podremos ver el estado de nuestros commits, en la primer parte veremos todos los archivos que han sido modificado, eliminados o agregados. En esta parte haremos clic en el boton verde “Stage all changes” para agregar los archivos ala zona de index y así poder hacer un commit.



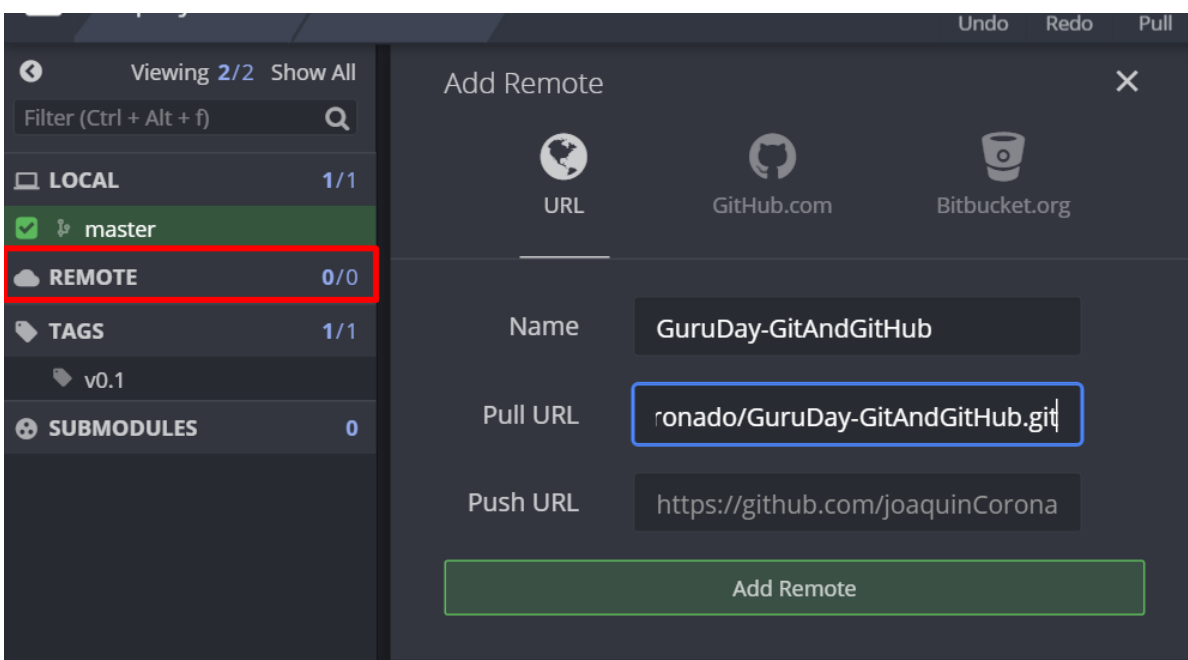
Enseguida se mostraran los archivos de nuestro proyecto en el área de index , en este momento podemos agregar la descripcion de nuestro commit y hacer clic en el boton verde “Commit”.



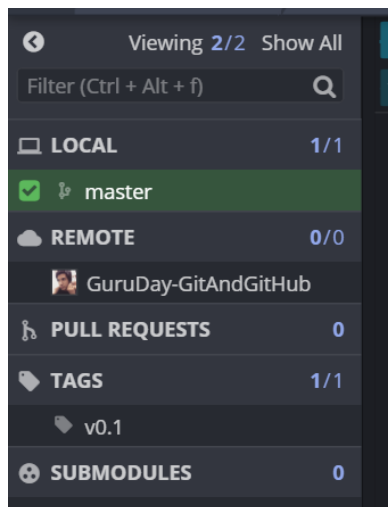
El commit creado lo podremos ver en la pantalla principal de GitKraken



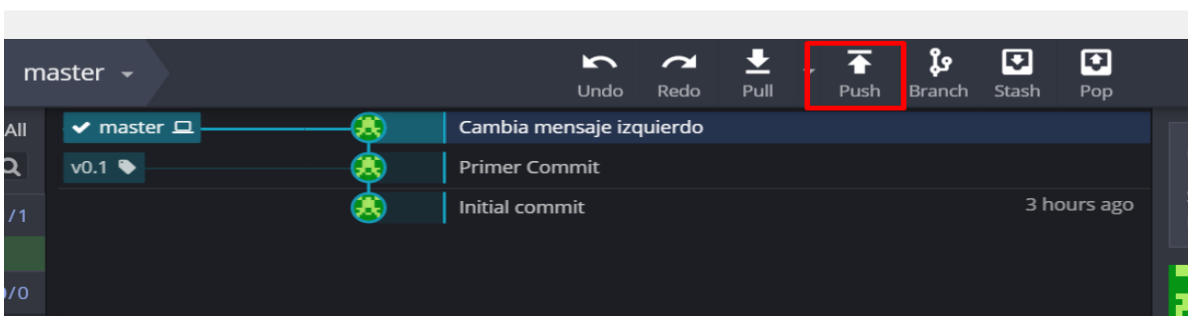
Para mandar nuestro repositorio a GitHub haremos clic en “REMOTE” y agregamos el nombre de nuestra conexión y la URL de nuestro repositorio.



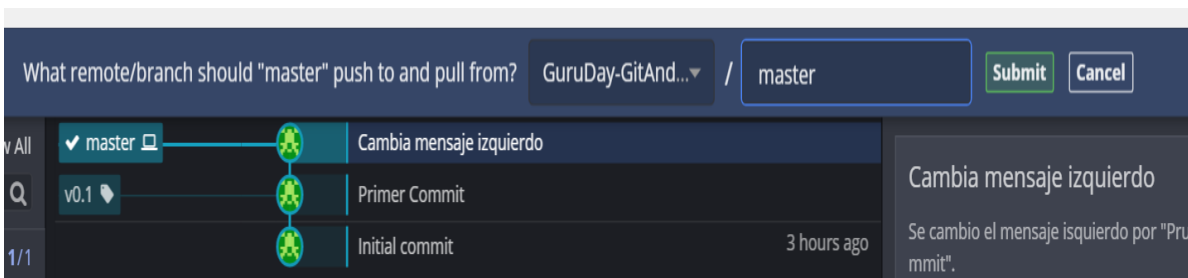
Una vez creada la conexión estamos listos para hacer nuestro primer “PUSH”.



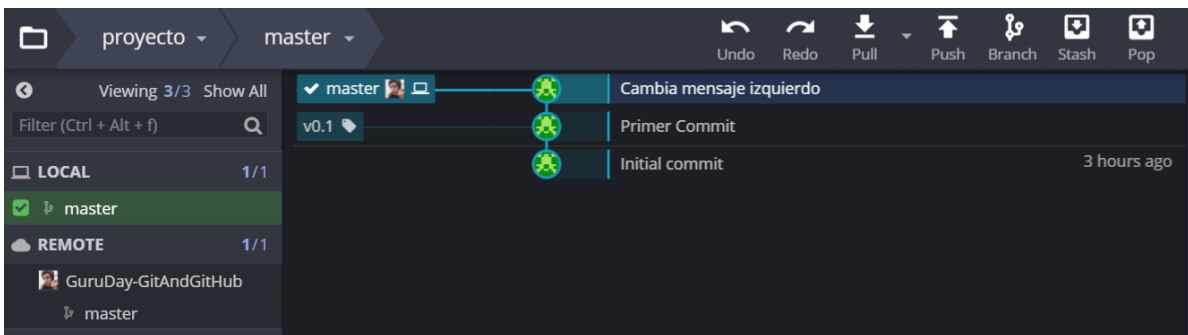
Haremos clic en el botón “PUSH” que se encuentra en la parte superior:



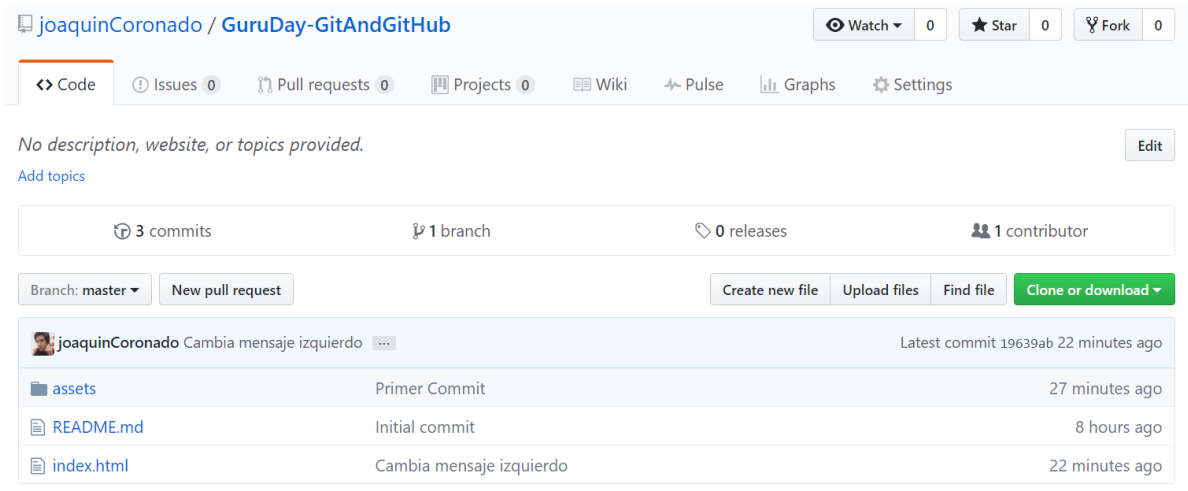
Enseguida seleccionamos la conexión que acabamos de crear y el nombre de la rama que vamos a subir GitHub:



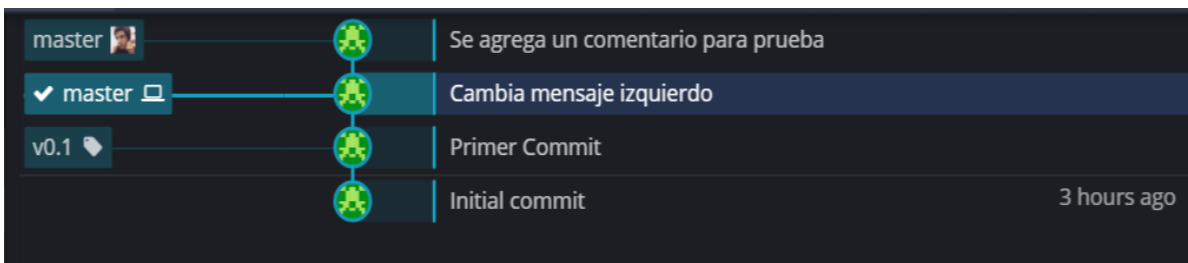
Una vez hecho el “PUSH” podremos notar que nuestra foto de perfil de GitHub se muestra en el commit al que hicimos “PUSH”



También verás que tu repositorio en GitHub se ha actualizado:



Podemos hacer un “PULL” para traer todos los cambios que se hicieron desde la nube y que nosotros no tenemos, nos damos cuenta de que existen cambios cuando nuestra rama “master” está por debajo de algún commit.



En este caso solo hacemos clic en el botón “PULL” que se encuentra en la parte superior:

