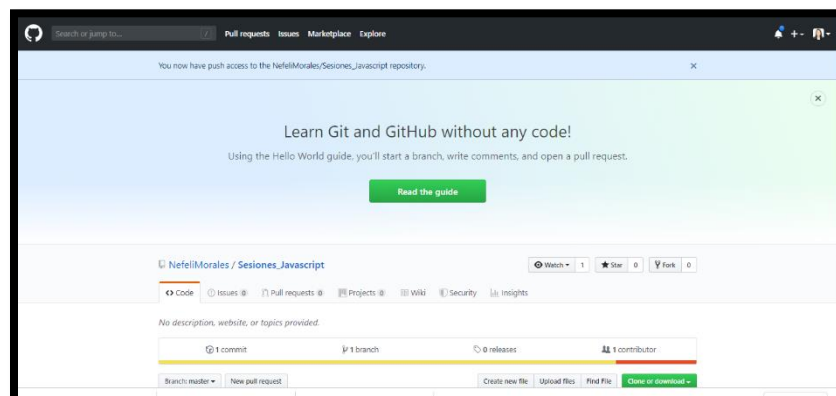


¿Cómo crear una cuenta en Github?

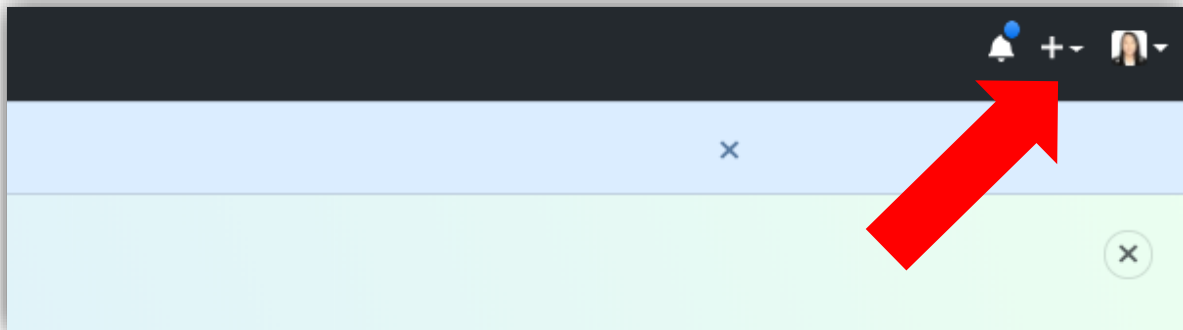
Para crear tu cuenta de **Github** debemos seguir los siguientes pasos:

1. Debemos de dirigirnos a la página oficial de **Github**: <https://github.com/>
2. Ya en la página, haremos clic en **Sign up**.
3. Ahora deberás escribir un **username**, que podrá ser uno que ya tengas. Por ejemplo, si tienes un **username** que creaste para un juego online podrías usar ese. No hay límites de cómo debe de ser, pero ten en cuenta que será visible para todo el mundo y puede que te lo pidan en algún momento para añadirte como colaborador de un proyecto, así que ten cuidado con lo que pongas como **username**. Luego ingresa tu **correo**, una **contraseña** y haz clic en **Create an account**.
4. En la siguiente pantalla asegúrate de marcar la opción **Unlimited public repositories for free**, esta es la versión "free" de **Github**; luego haz clic en **continue**.
5. El paso 3 del registro puedes completarlo con tus datos y hacer clic en **Submit**, o solo hacer clic en **Skip this step**.

¡Y listo! Al hacer todos estos pasos ya tendríamos nuestra cuenta de **Github** creada.



¿Cómo crear un Repositorio en github?



Create a new repository

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

Owner: NefeliMorales / Repository name:

Great repository names are short and memorable. Need inspiration? How about [ideal-chainsaw](#)?

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.

Add .gitignore: None | Add a license: None ⓘ


Quick setup — if you've done this kind of thing before

 Set up in Desktop or ☐ HTTPS ☐ SSH `https://github.com/NefeliMorales/Sesiones_Javascript.git` 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Sesiones_Javascript" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/NefeliMorales/Sesiones_Javascript.git
git push -u origin master
```



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

```
git remote add origin https://github.com/NefeliMorales/Sesiones_Javascript.git
git push -u origin master
```



...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

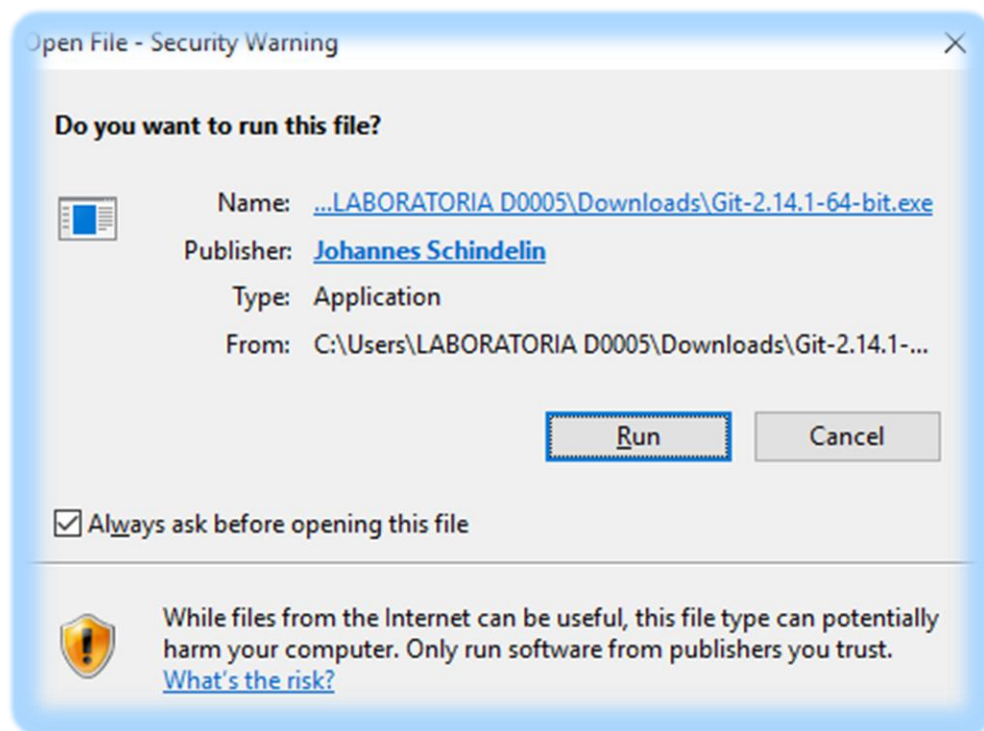
¿CÓMO INSTALAMOS GIT?

Si nos encontramos en Linux o Mac OS X no será necesario hacer la instalación de Git porque ya viene instalado, pero si nuestro sistema operativo es **Windows** debemos descargar e instalar **Git** en nuestro equipo.

- Descarga Git <https://git-scm.com>

Cuando haya terminado la descarga de nuestro programa comenzaremos con la instalación haciendo doble clic en el archivo que se acaba de descargar y luego nos aparecerán las siguientes pantallas:

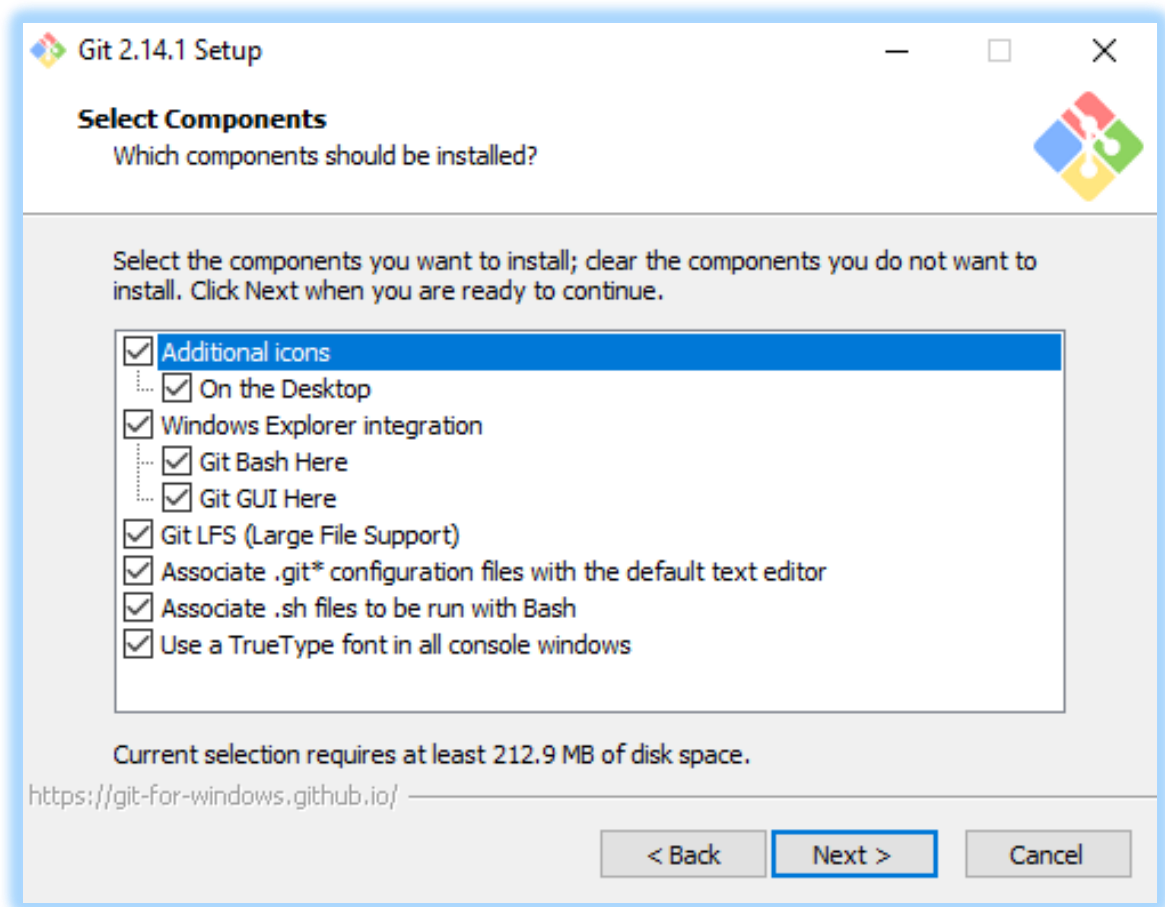
1. Hacemos clic en el botón **Run**



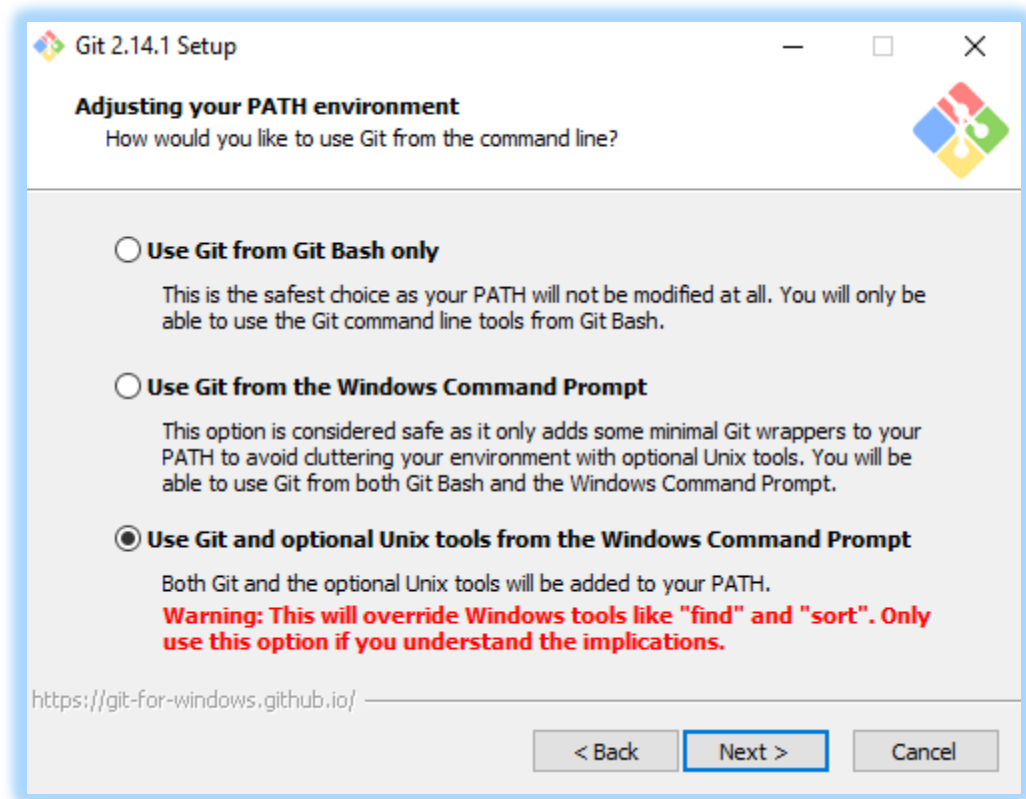
2. Ahora en **Next >**



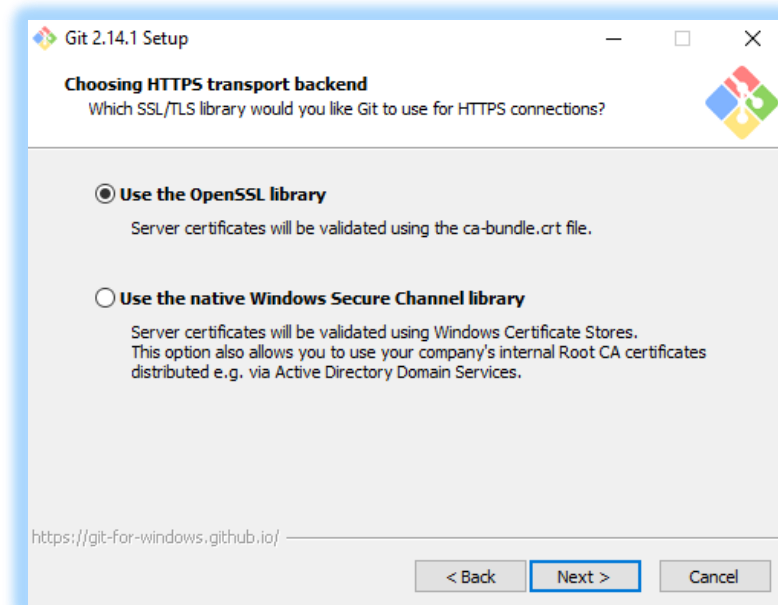
3. Nos aparecerá la siguiente pantalla, aquí seleccionaremos las siguientes opciones y hacemos clic en **Next >**



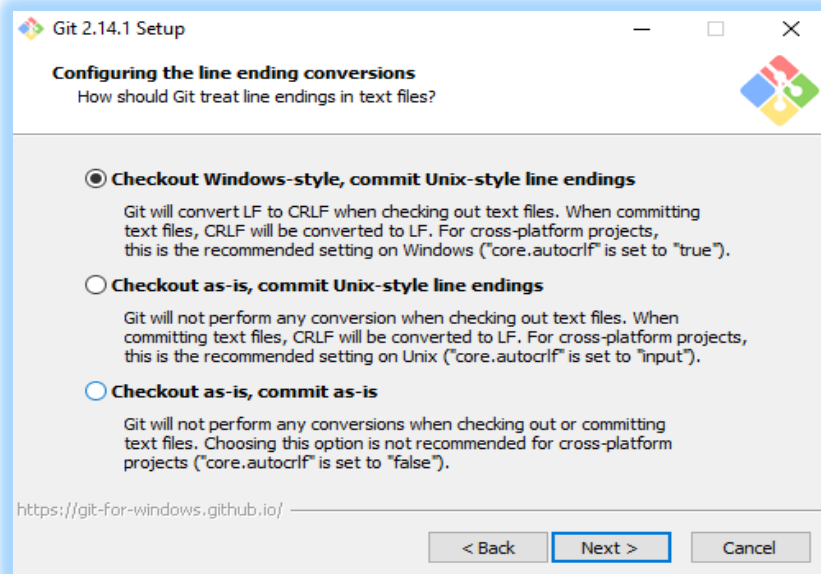
4. Seleccionamos la opción **Use Git and optional Unix tools from the Windows Command Prompt** y hacemos clic en **Next >**



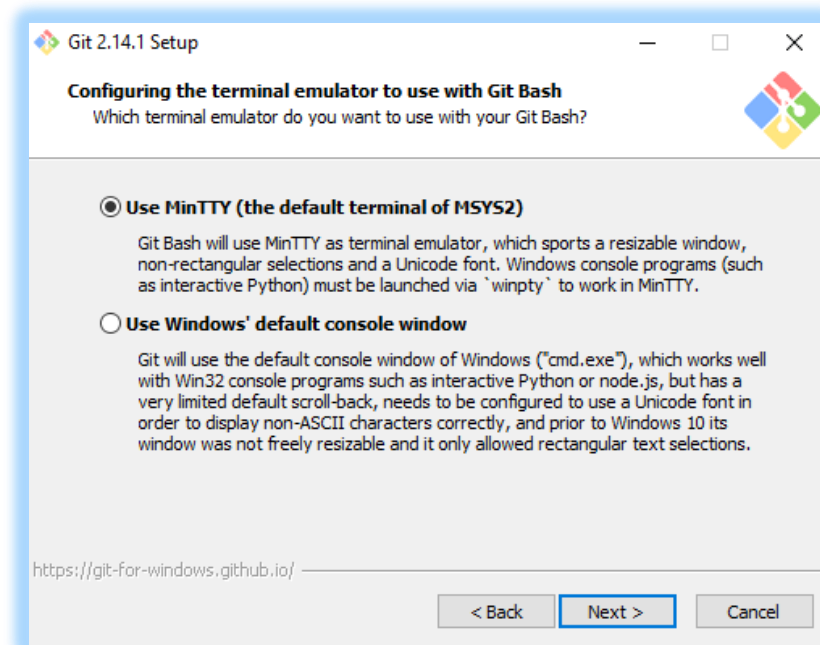
5. Seleccionamos la opción **Use the OpenSSL library** y continuamos (**Next >**)



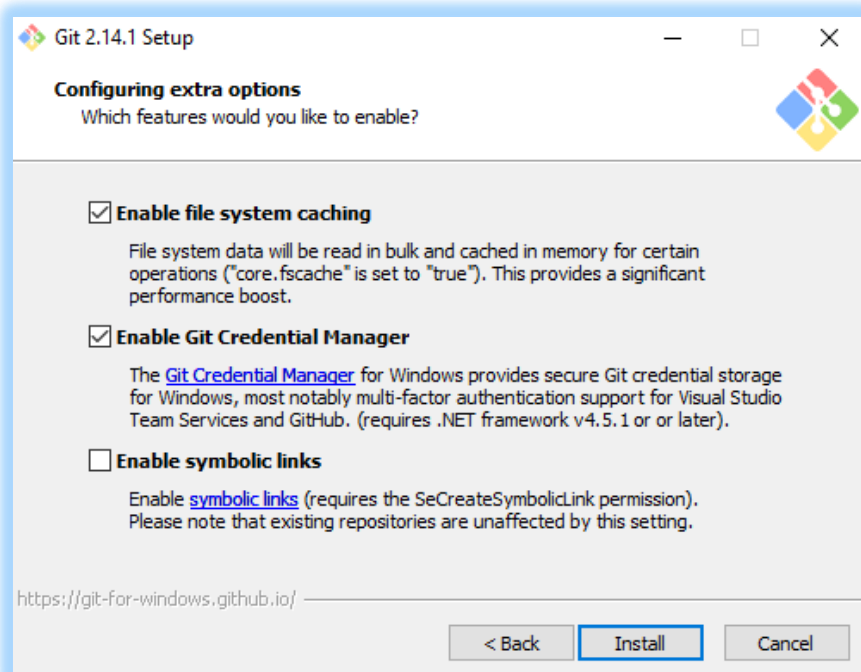
6. Elegimos la primera opción y seguimos



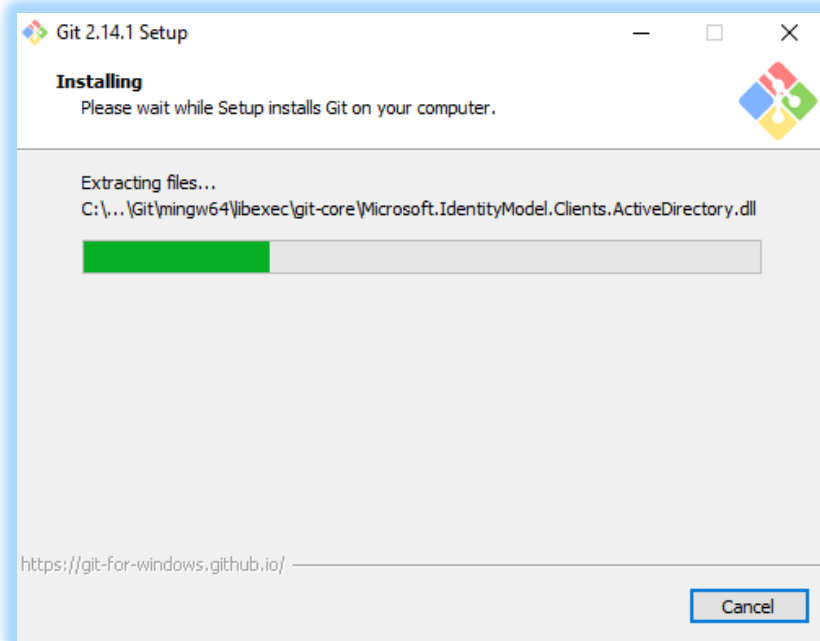
7. Nos aseguramos de escoger la opción **Use MinTTY**



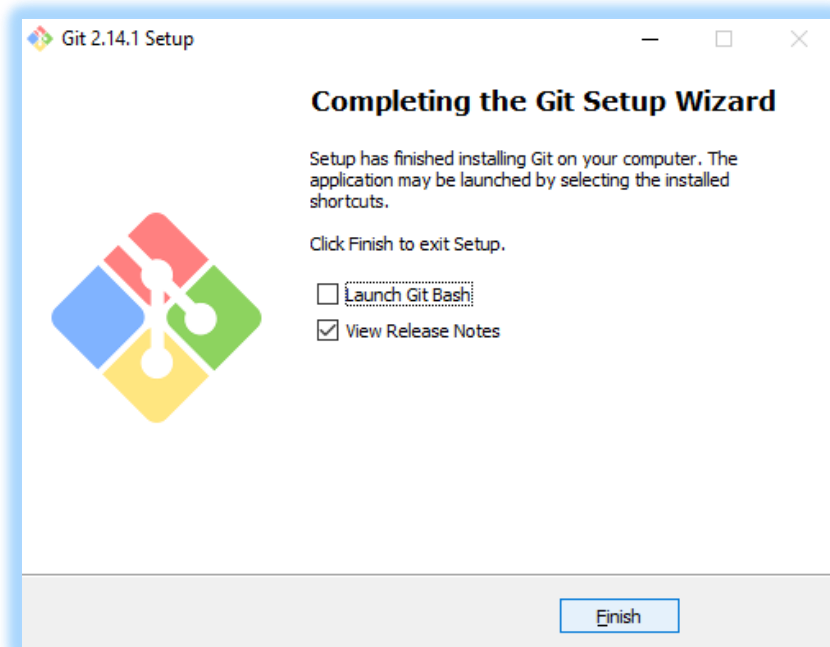
8. Seleccionamos las dos primeras opciones y hacemos clic en **Install**



9. Ahora comenzará a instalarse



10. Y ya tendremos instalado **Git** en nuestro equipo al hacer clic en **Finish**



¿CÓMO CONFIGURAMOS GIT?

Ya tenemos instalado `Git` en nuestro equipo, ahora personalizaremos nuestro entorno de `Git`.

Recuerda que si quieres ejecutar comandos de Git y te encuentras en Mac OS X o Linux deberás ejecutarlos en **la línea de comandos**, en cambio, si tu sistema operativo es Windows abriremos **Git Bash** para ejecutar los comandos de Git.

Ahora nosotras vamos configurar nuestro entorno de `Git`:

- Para configurar nuestro **username** debemos escribir la siguiente línea que debe de tener nuestros datos, y luego presionaremos la tecla "enter":

```
git config --global user.name Tu-username
```

- Para configurar nuestro **correo electrónico** debemos escribir la siguiente línea con nuestros datos y luego presionaremos la tecla "enter":

```
git config --global user.email tucorreo@gmail.com
```

Los datos con los que llenes las líneas anteriores deben de ser los mismos que usaste al crear tu cuenta de `Github`, porque como te comentaba con ellos se registrarán los `commits` que hagas en `Git` y que luego pasarán al historial de nuestro proyecto en `Github`.

¿Cómo usar Git y Github?

Hemos leído las ventajas que nos pueden brindar **Git** y **Github**, con ellos podremos subir un repositorio local a uno remoto, trabajar con versiones y colaborativamente, etc. Ahora hagamos la "magia" de trabajar con **Git** y **Github**, vamos a subir nuestro **repositorio local** a uno **remoto** y trabajar colaborativamente.

Detalle de comandos

A continuación veremos una lista de los comandos de **git** usados en esta lección:

Comando	Descripción
<code>git config --global user.name Tu-username</code>	Configura el nombre de usuario
<code>git config --global user.email tucorreo@gmail.com</code>	Configura el correo electrónico
<code>git init</code>	Inicia el seguimiento a una carpeta indicada
<code>git add archivo.extensión</code>	Indica el archivo que pasarán del <i>working directory</i> al <i>staging area</i>
<code>git add .</code>	Indica que todos los archivos pasarán del <i>working directory</i> al <i>staging area</i>
<code>git commit -m 'añadiendo un comentario'</code>	Confirma los archivos preparados anteriormente con <code>git add</code>
<code>git remote add nombre-remoto url</code>	Vincula el repositorio local con el repositorio que se encuentra en la "nube" por medio de la url
<code>git push nombre-remoto master</code>	Envía la nueva versión a la rama <i>master</i> del repositorio remoto
<code>git status</code>	Indica el estado de los archivos del repositorio local
<code>git pull nombre-remoto master</code>	Actualiza el repositorio local al último commit que se tenga registrado en el repositorio remoto

A continuación, tienes la lista de los comandos más comunes que utilizarás trabajando con **Git** y **Github**.

- **init**: Este comando le indica a **git** que comenzará a hacer seguimiento de la carpeta actual. En otras palabras, comenzará a vigilar esta carpeta. El comando para iniciar el seguimiento de la carpeta sería:

```
$ git init
```

Al ejecutar esta línea, **git** creará una carpeta oculta llamada **.git** en el working directory, en la raíz del proyecto.

- **add**: Con este comando estaremos indicando qué archivos queremos que pasen de nuestro *working directory* al *staging area*. Es decir que los archivos ya han sido modificados y están **preparados** para la nueva versión del proyecto.

Por ejemplo, si queremos añadir a nuestro *staging area* el archivo **HTML** llamado **index.html** debemos de ejecutar la siguiente línea:

```
$ git add index.html
```

Si queremos que todos los archivos del directorio actual se añadan al *staging area* (definir que ya están preparados), ejecutamos la siguiente línea:

```
$ git add .
```

- **commit**: Con **commit** **confirmaremos** los archivos que declaramos como preparados con el comando anterior (**add**). Con los archivos confirmados se creará una nueva versión del proyecto.

La estructura del comando es la siguiente:

```
$ git commit -m 'añadiendo un comentario'
```

El texto que va dentro de las comillas (") puede ser cualquiera, te recomiendo que escribas un resumen de lo que estás subiendo en ese momento para que tengas una mejor organización, además, un buen comentario te servirá si quieres volver a una funcionalidad en especial sin tener que leer todo el código y solo guiarte por el comentario de tu commit. Por ejemplo:

```
$ git commit -m 'Creando estructura HTML'
```

- **remote**: Con **remote** **vincularemos** nuestro repositorio local con nuestro repositorio que se encuentra en la "nube" por medio de la **url**. Este paso no se debe de hacer cada vez que creamos una nueva versión, solo cuando queramos especificar el repositorio remoto que tendremos. Entonces, para enlazar nuestro repositorio local con nuestro repositorio remoto debemos de ejecutar el siguiente comando:

```
$ git remote add nombre-remoto url
```

Normalmente, la mayoría de los developers, como **nombre-remoto** utiliza **origin** y **url** es la "ruta" (url) de nuestro repositorio en **Github**. Por ejemplo, hace un momento he creado el repositorio "prueba" en **Github** y si quiero vincular mi repositorio local con ese repositorio remoto tendría que ejecutar la siguiente línea de comando:

```
git remote add origin https://github.com/NefeliMorales/Sesiones_Javascript.git
```

- `push` : Por medio de `push` **enviaremos** nuestra nueva versión que confirmamos en el `commit` a nuestro repositorio remoto. El comando es el siguiente:

```
$ git push nombre-remoto master
```

En **nombre-remoto** pondremos el mismo nombre-remoto que le hemos asignado al momento de añadir la url de nuestro proyecto (en nuestro caso sería **origin**) y seguido pondremos **master**, más adelante conoceremos el trabajo en "ramas" que tiene `Github`. Las **ramas** son una copia paralela que podemos crear de nuestro código. Por defecto, los repositorios traen una "rama" llamada **master** y ahora trabajaremos solo en ella, así que subiremos nuestros cambios a ella especificando **master** como el nombre de la rama a la que queremos subir nuestra versión, entonces el comando a ejecutar sería:

```
$ git push origin master
```

- `status` : Con `status` conoceremos el estado de los archivos de nuestro repositorio local. Es decir, nos dirá si existe un archivo que se ha modificado y no se ha actualizado en el repositorio remoto o si hay un archivo que todavía no se "sube" y nos informará si se encuentra en nuestro working directoy o staging area. Este comando nos resultará muy útil para determinar qué archivos debemos declarar como "preparados" (por medio del comando `add`) y luego "confirmarlos" en la nueva versión (por medio del comando `commit`) que subiremos a nuestro repositorio remoto. El comando para ver el estado de nuestro repositorio local es:

```
$ git status
```

- `pull`: Sirve para actualizar nuestro repositorio local al último `commit` que tengamos registrado en el repositorio remoto. Este comando nos servirá mucho al momento de trabajar colaborativamente porque podremos tener los avances que nuestros colaboradores han subido al repositorio remoto a nuestro repositorio local y mantenerlo actualizado. El comando es el siguiente:

```
$ git pull nombre-remoto master
```

En nuestro caso, como nuestro **nombre-remoto** ya lo declaramos como **origin** y la rama en la que estamos trabajando se llama **master**, el comando sería:

```
$ git pull origin master
```

<p>Si se puede
imaginar, se puede
programar</p>