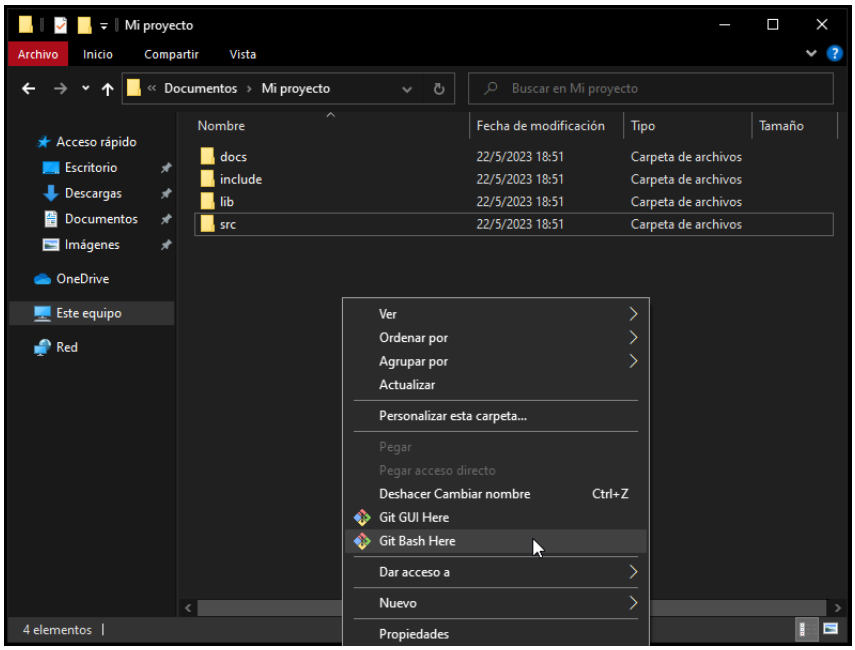
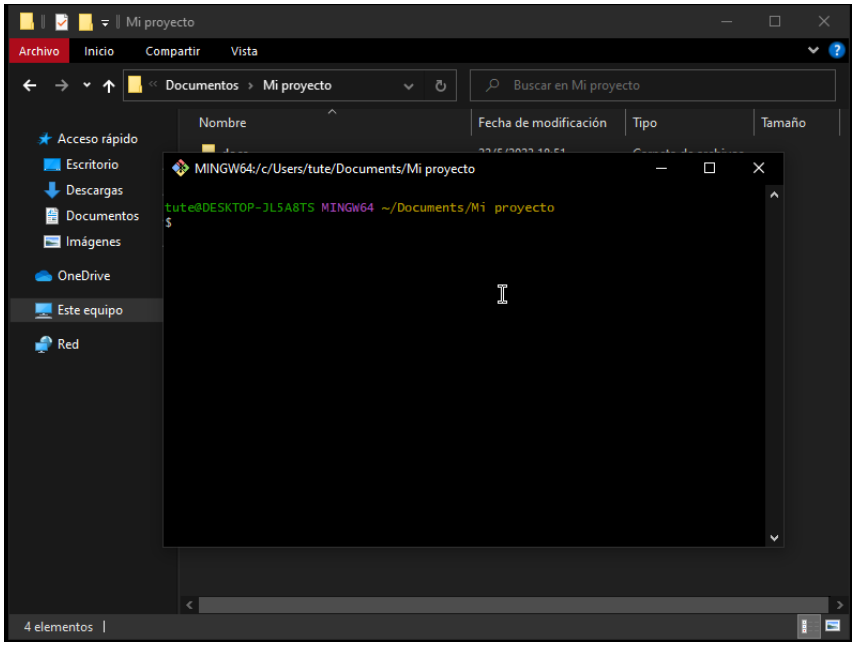
Para inicializar un proyecto con git y subirlo a un repositorio remoto, primero debemos configurar git (solamente la primera vez) y luego seguir los siguiente pasos:

1. Agregar un archivo `.gitignore`.
2. Inicializar el repositorio. (`git init`)
3. Agregar los archivos que formarán parte de nuestro repositorio. (`git add`)
4. Asentar los cambios en el repositorio local. (`git commit`)
5. Crear un repositorio remoto donde alojar nuestro proyecto. (`git remote`)
6. Enviar los cambios al repositorio remoto. (`git push`)

0. Configurar el usuario de `git`

Antes de empezar iremos a la carpeta de nuestro proyecto y abriremos una terminal allí, en el caso de MS Windows puede hacer clic con el derecho en la carpeta correspondiente e iniciar `Git Bash`.

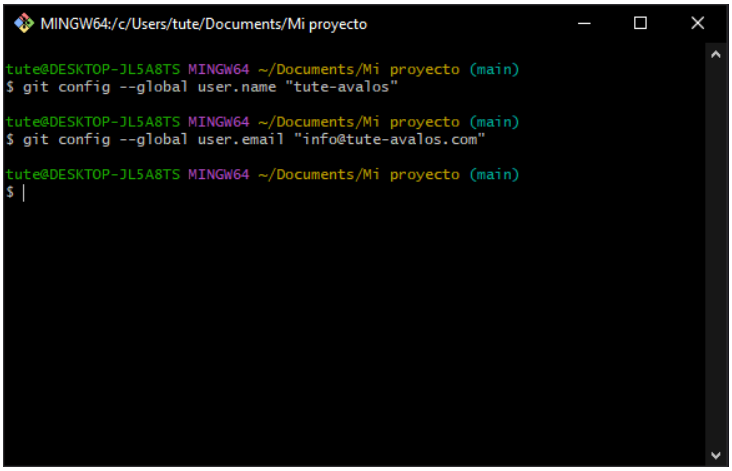




Una vez abierta la terminal, configuraremos el usuario que utilizaremos para trabajar. Este es el que aparecerá como responsable de los cambios al momento de confirmar cambios en el repositorio.

Debemos utilizar el sub-comando `config` para modificar los atributos `name` y `email` de la variable `user`.

```
$ git config --global user.name  "nombre-usuario"
$ git config --global user.email "user@email.com"
```

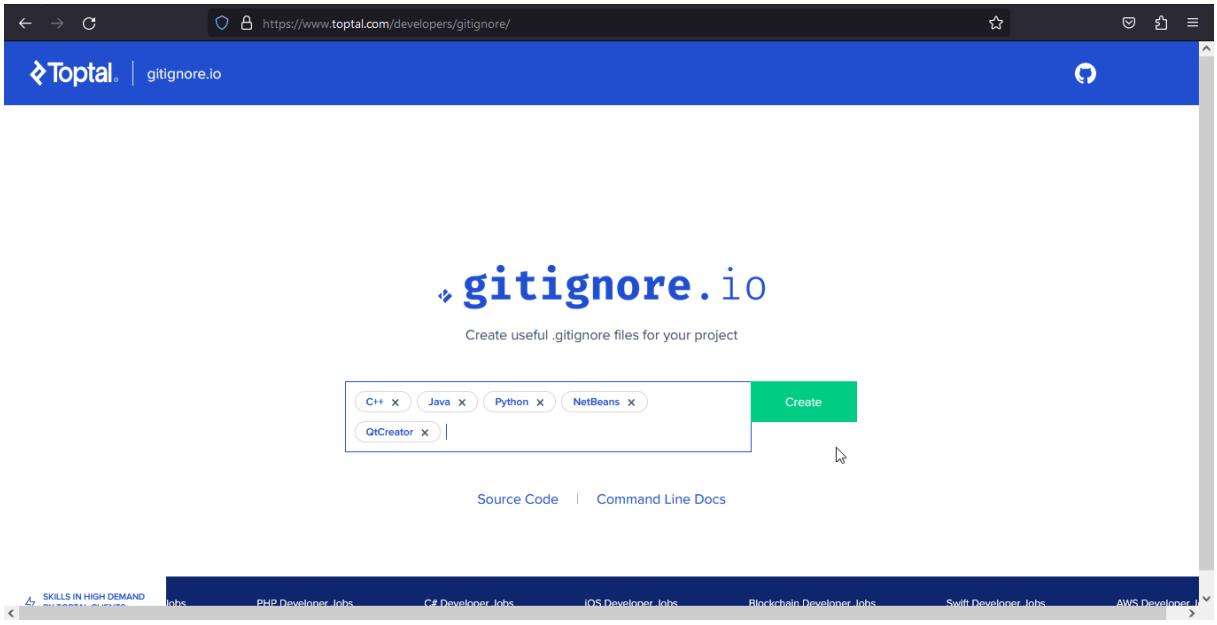


Esto configura el usuario y correo de forma global, si quisiera utilizar otro usuario o email en algún repositorio, puede hacerlo sacando el parámetro `--global`.

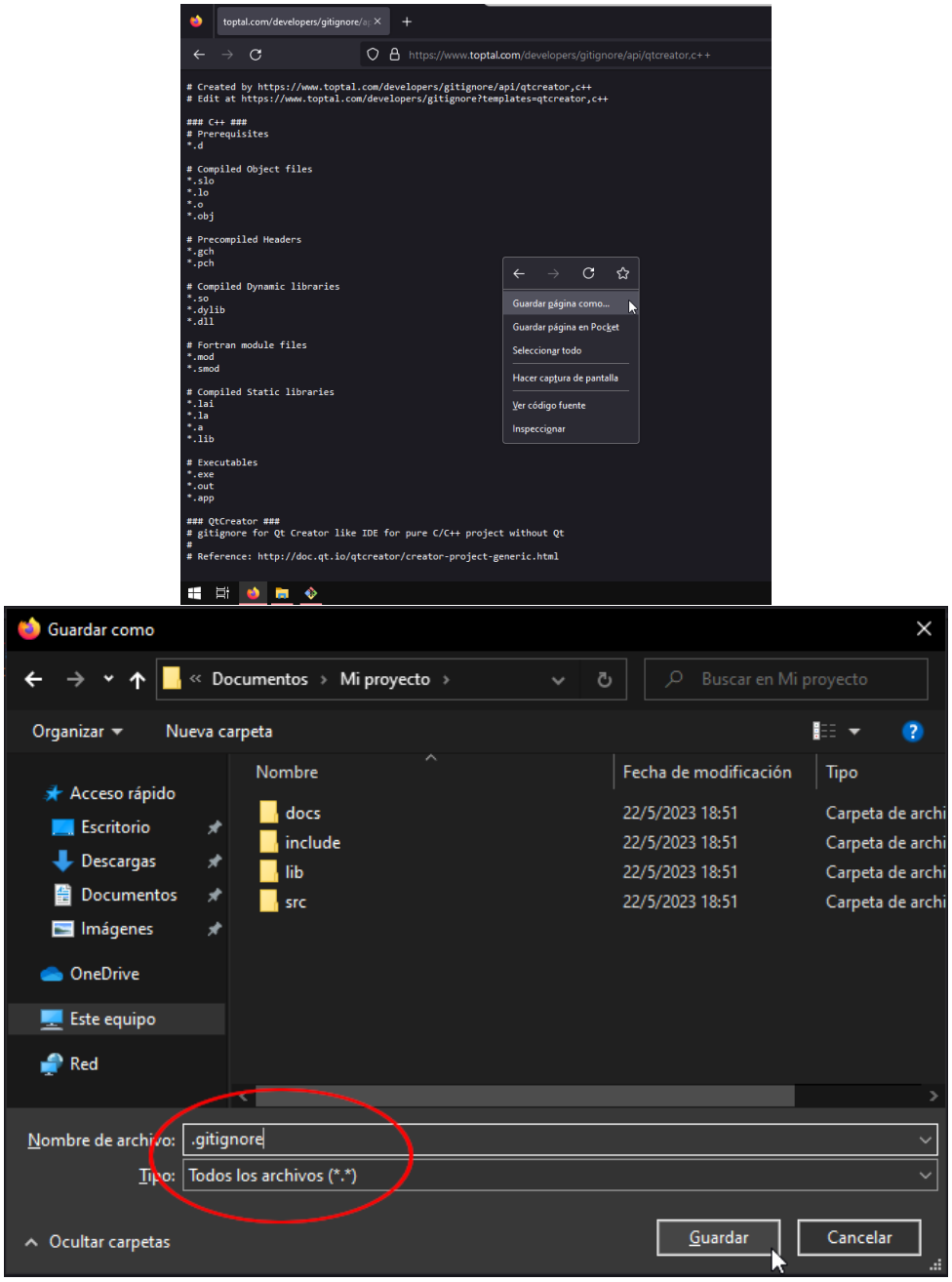
1. Agregar un `.gitignore`

Es **altamente recomendable** agregar un archivo `.gitignore` antes de inicializar el repositorio. Esto evitará que archivos o carpetas que no forman parte del código fuente o que no deban ser considerados necesarios para utilizar el repositorio formen parte del mismo. Esto es aplicable a archivos compilados, binarios, cache, temporales, etc.

Esto varía mucho entre lenguajes y herramientas de desarrollo que se utilizan habitualmente. Es por ello que la mejor manera de confeccionar este archivo es utilizando una herramienta online: `https://gitignore.io/`. Ésta generará el archivo correspondiente dependiendo de los lenguajes, IDEs o herramientas que utilicemos en nuestro proyecto.

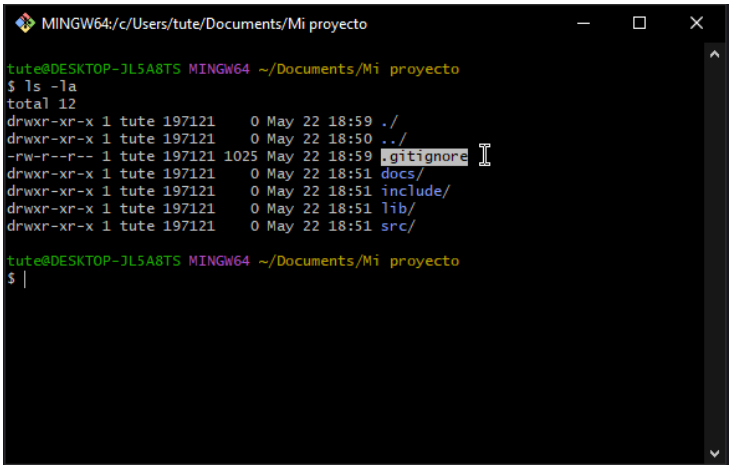


Una vez que se hace clic en `Create`, se generará un texto plano el cual debemos guardar con el nombre `.gitignore` dentro de la raíz de nuestro proyecto. En MS Windows debemos prestar atención de guardarlo sin un formato especifico.



Podemos asegurarnos de que agregamos el `.gitignore` correctamente listando los archivos en la carpeta con el comando `ls`.

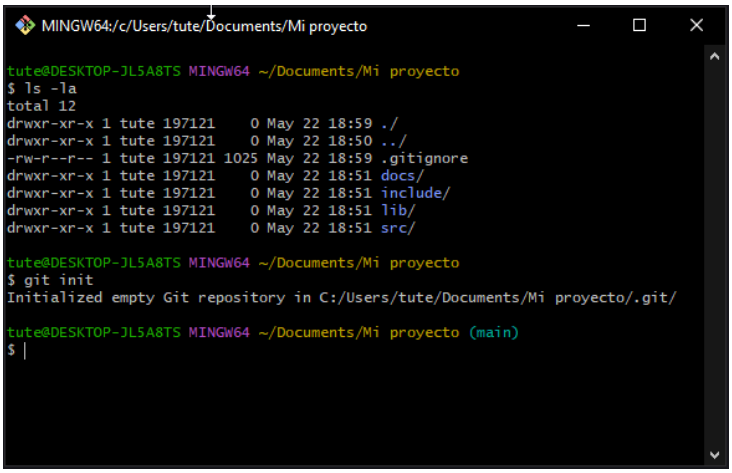
```
$ ls -la
```



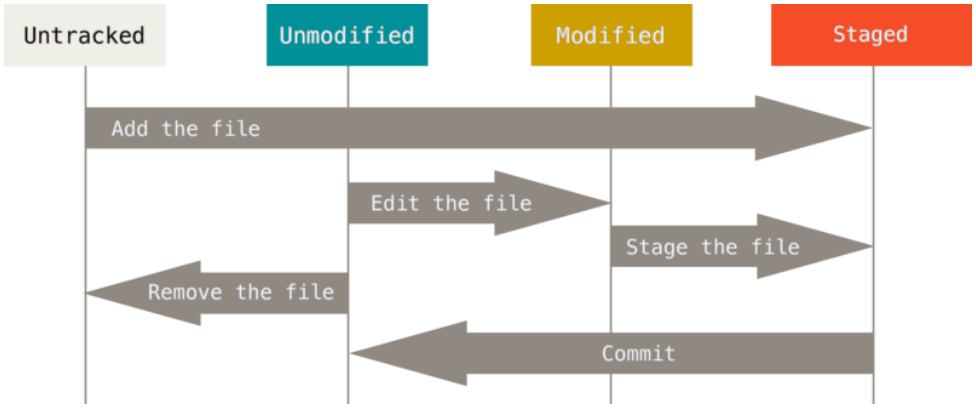
2. Inicializar el repositorio

Una vez incluido el `.gitignore` se puede inicializar el repositorio con seguridad. Para ello se utiliza el sub-comando `init`. Esto creará la rama por defecto `main`, la única presente por el momento en nuestro repositorio.

```
$ git init
```

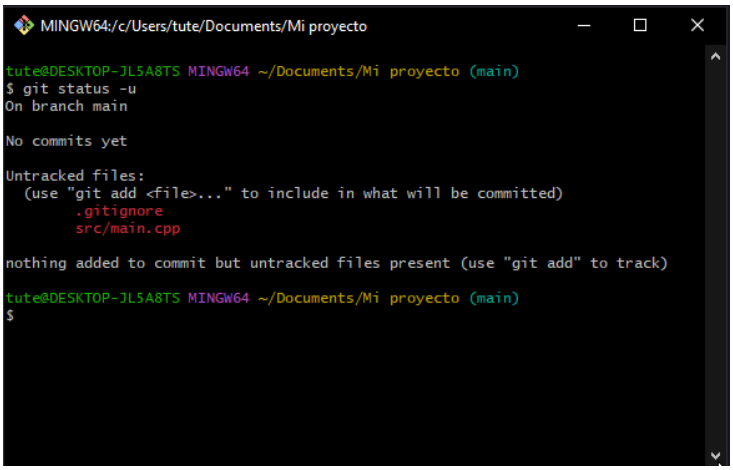


Una vez inicializado, los archivos tiene dos estados posibles `tracked` (rastreado) y `untracked` (no rastreado). Los que están rastreados son los que fueron agregados y Git sabe que existen y tiene la noción de los 3 estados posibles en los que se encuentran: `unmodified`, `modified` o `staged`.



Los archivos en un repositorio recién inicializados están todos en el estado **no rastreado**. Esto podemos saberlo al invocar el sub-comando `status`.

```
$ git status -u
```



Nota

Preste atención que Git **no hara seguimientos de carpetas vacías**, si no hay contenido en las mismas no hará el *tracking*. Si desea que una carpeta forme parte de su repositorio, deberá agregar al menos un archivo en ella.

Para más información sobre este tema puede consultar `https://git-scm.com/book/es/v2/Fundamentos-de-Git-Guardando-cambios-en-el-Repositorio`

3. Agregar archivos

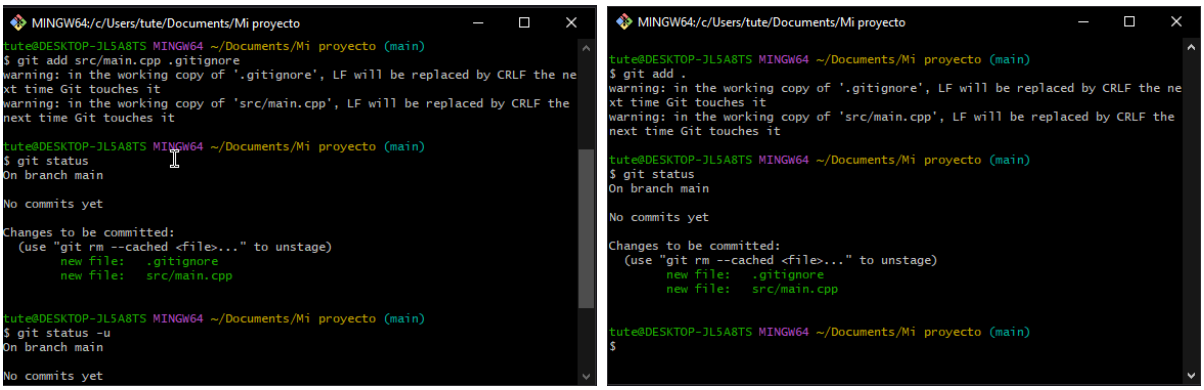
En Git existen diferentes estados en los cuales se encuentran los archivos de nuestro proyecto (además de `tracked` y `untracked`). Donde nos encontramos habitualmente es en lo que se denomina `Working Directory/Workspace` (espacio de trabajo). Allí estaremos siempre haciendo cambios o agregando archivos nuevos a nuestro proyecto.

Podemos verificar, como se demostró, el estado de los archivos de nuestro proyecto con el sub-comando `status`, el cual nos mostrará los archivos nuevos, modificados, sin cambios y borrados. Una vez que estamos conformes con el estado actual movemos los archivos al estado `staged` a través del sub-comando `add` listando todos los archivos que queremos agregar a este estado.

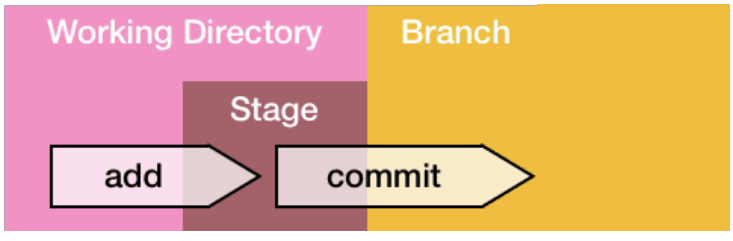
```
$ git add <archivo1> <archivo2> ... <archivoN>
```

Si queremos agregar todos los archivos al *stagin* podemos utilizar la abreviatura `.`:

```
$ git add .
```



Pero se confirmarán los cambios recién al hacer la confirmación para formar parte del repositorio local en la rama correspondiente (en este caso `main`).

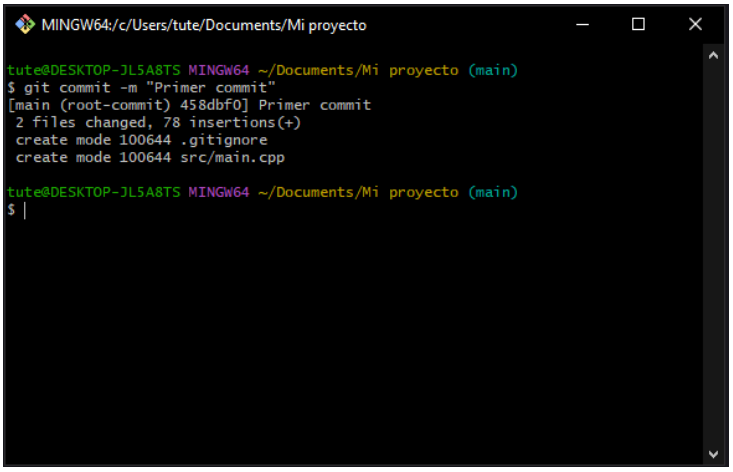


4. Asentar cambios

Una vez que estén los archivos en el `stage`, podemos confirmar o asentar los cambios en la rama con el sub-comando `commit`. Lo cual establece un punto fijo en el repositorio, al cual será posible volver o crear ramas derivadas del mismo.

```
$ git commit -m <Mensaje del Commit>
```

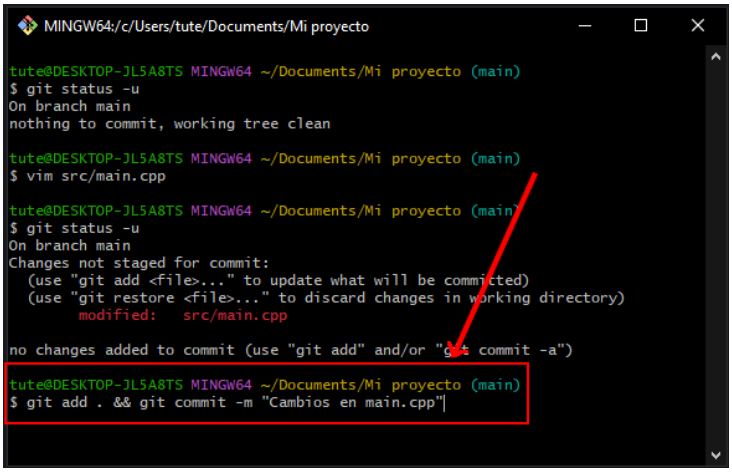
Es muy importante agregar un comentario descriptivo sobre el commit, ya que será de utilidad si se debe volver a un punto anterior, esta será muchas veces la única información útil para ello.



A partir de aquí, si solo se mantuviera una rama, el flujo de trabajo es el mismo. Se hacen modificaciones y se agregan al `stage` y se *commitean*, creando nuevas etapas a las cuales se puede volver.

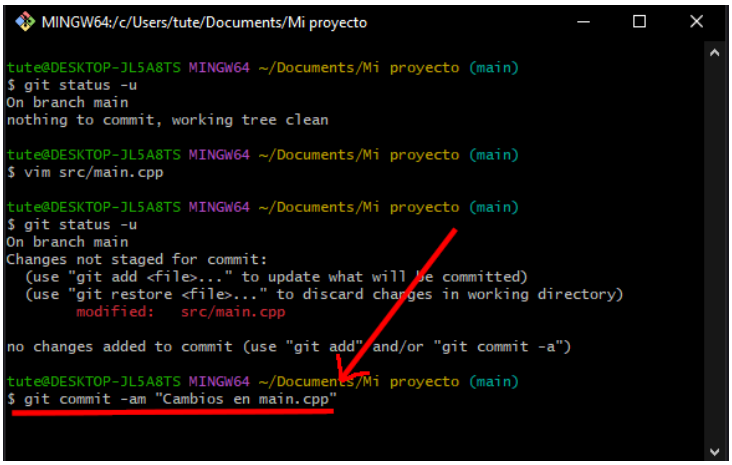
Si se modificara o agregara un archivo, estos deberán ser agregados con el sub-comando `add` y luego utilizar `commit` para ser confirmados.

```
$ git add <archivoNuevo> <archivoModificado> # o simplemente .
$ git commit -m <Mensaje del nuevo commit>
```



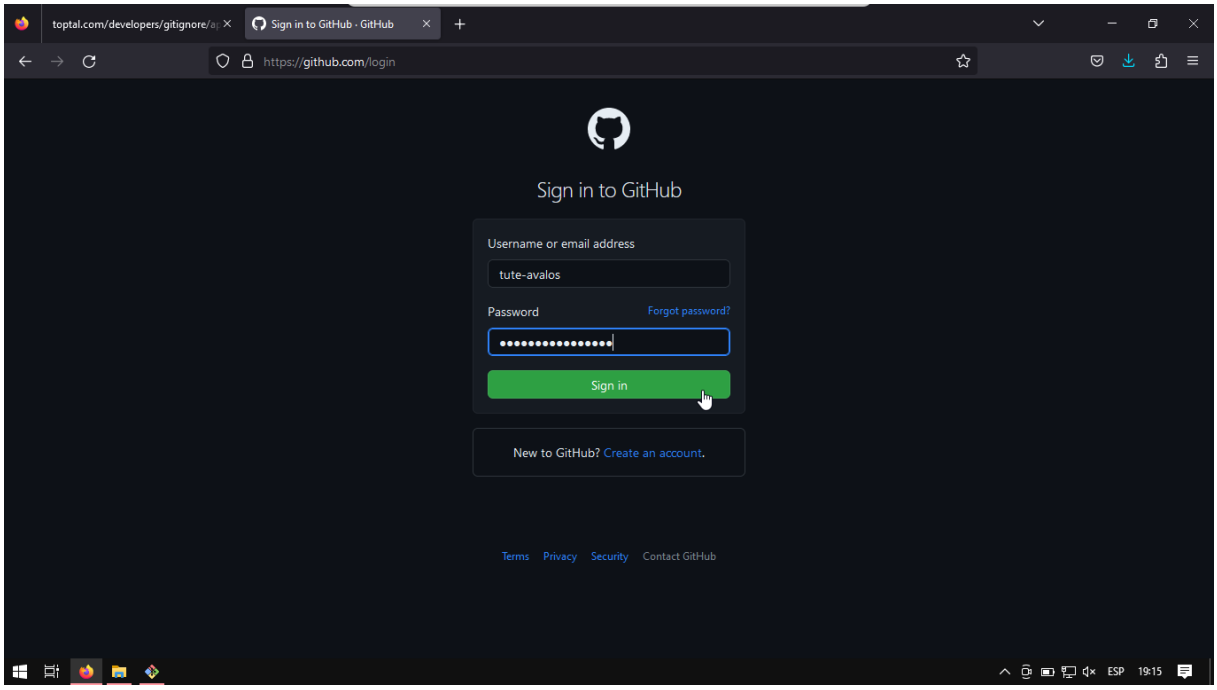
En el caso de que no se agregaran nuevos documentos, es decir que solo hay archivos marcados como *modificados*, entonces se puede ahorrar el sub-comando `add` utilizando únicamente `commit`:

```
$ git commit -am <Mensaje del nuevo commit>
```

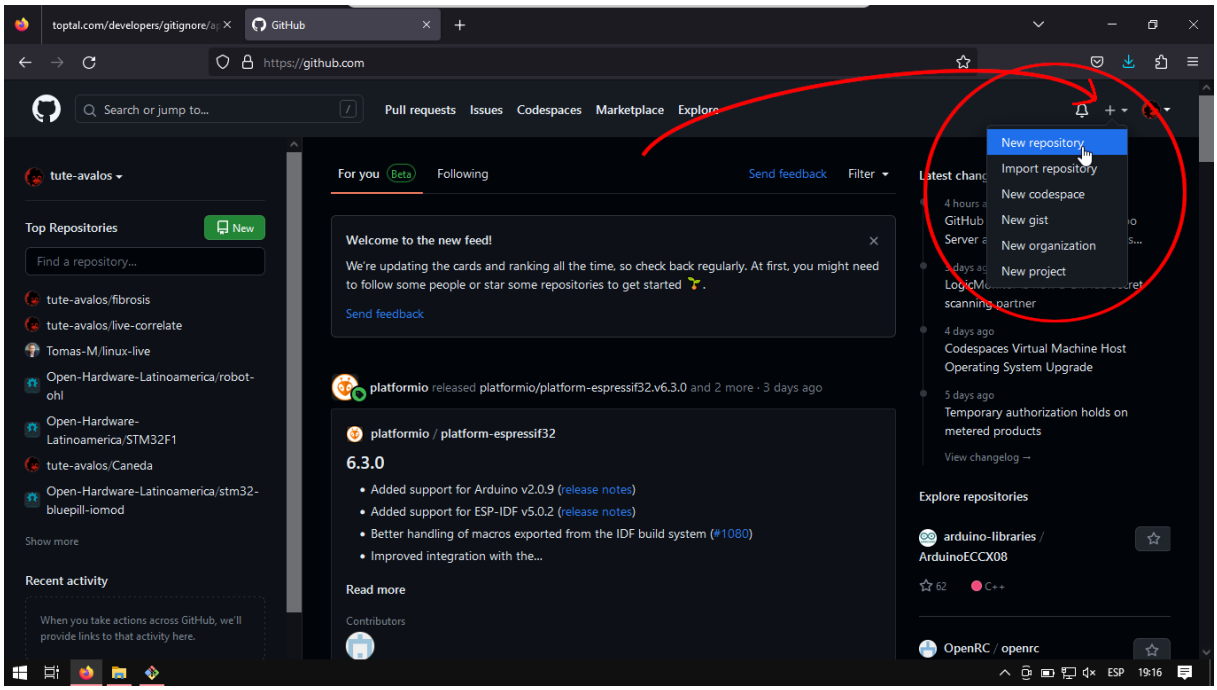


5. Crear un repositorio remoto

Para poder *guardar* nuestro repositorio en un servidor, para compartirlo o trabajar de forma remota desde cualquier estación de trabajo, entonces podemos utilizar un servicio como son GitLab, GitHub, Bitbucket, etc. los cuales cuentan con paquetes *freemium*. En este caso, se expondrá como crear un repositorio remoto en `https://github.com/`, para lo cual deberá tener cuenta.



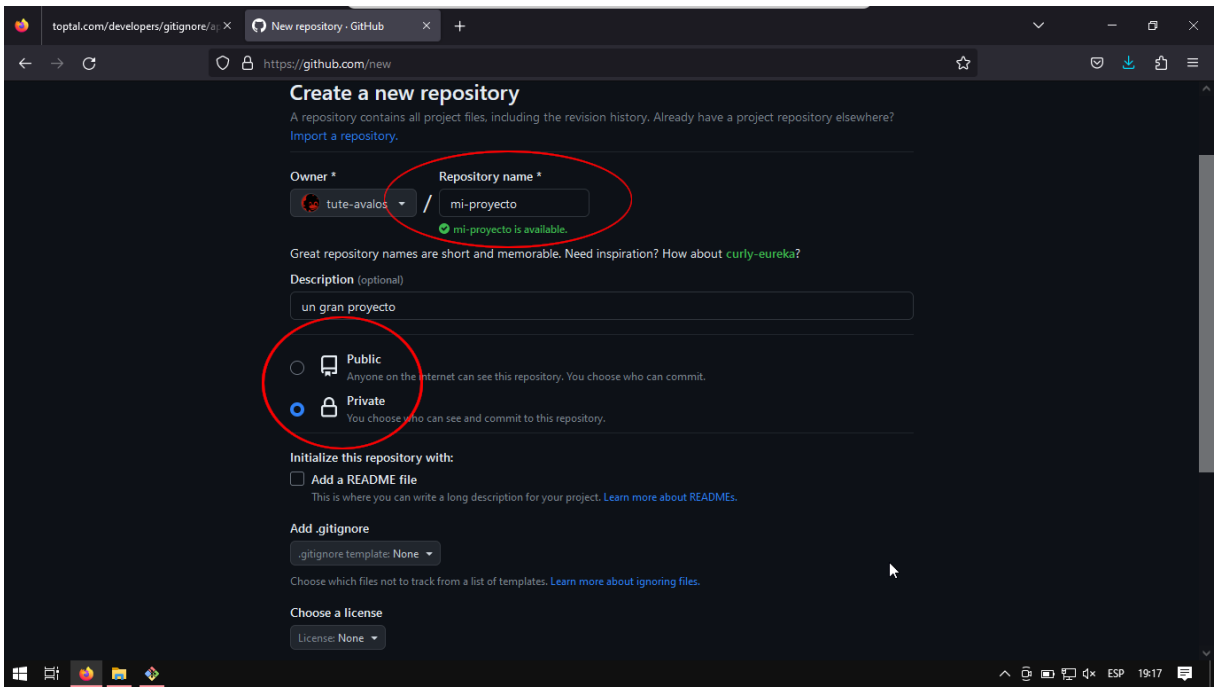
Una vez allí, se creará un nuevo repositorio haciendo clic en el '+' ubicado en la esquina superior derecha y luego en `New Repository`.



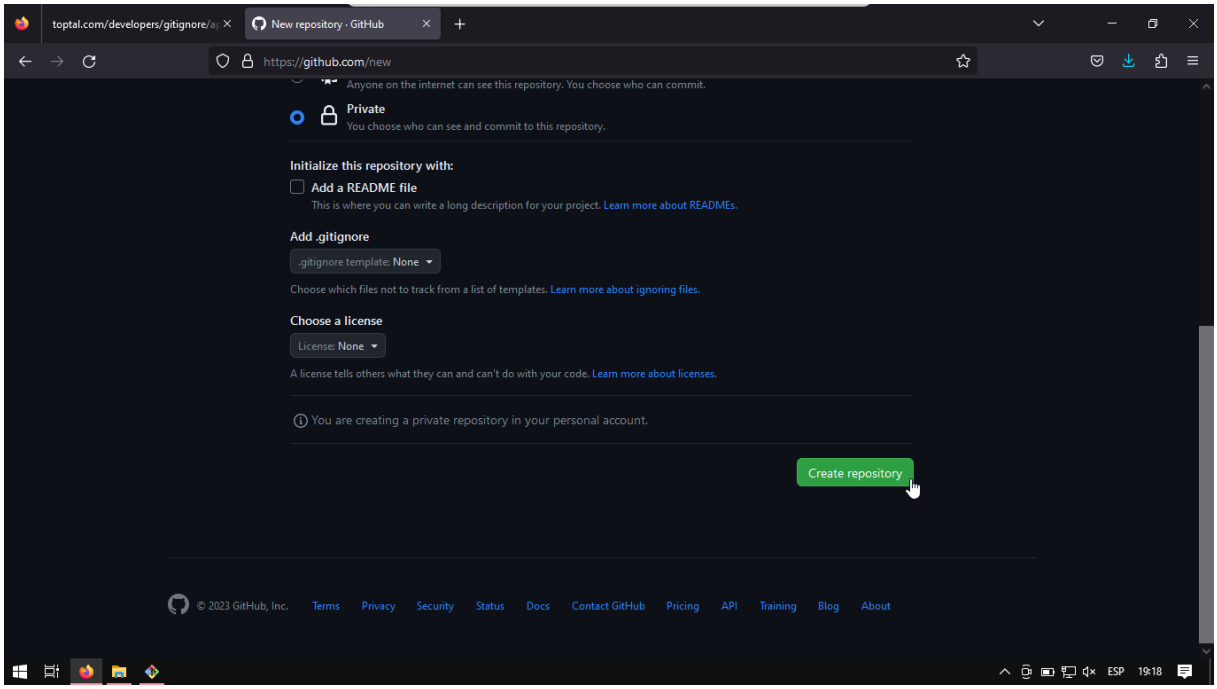
Eso nos llevará a un formulario donde debemos completar el nombre de nuestro proyecto, podremos agregar una descripción (opcional) y la visibilidad que queremos que tenga. Aquí podemos determinar quienes tienen acceso a nuestro repositorio siendo:

Public Cualquiera puede clonar y ver el repositorio, proponer cambios, etc.

Private Solo los usuarios que nosotros invitemos pueden acceder al repositorio.

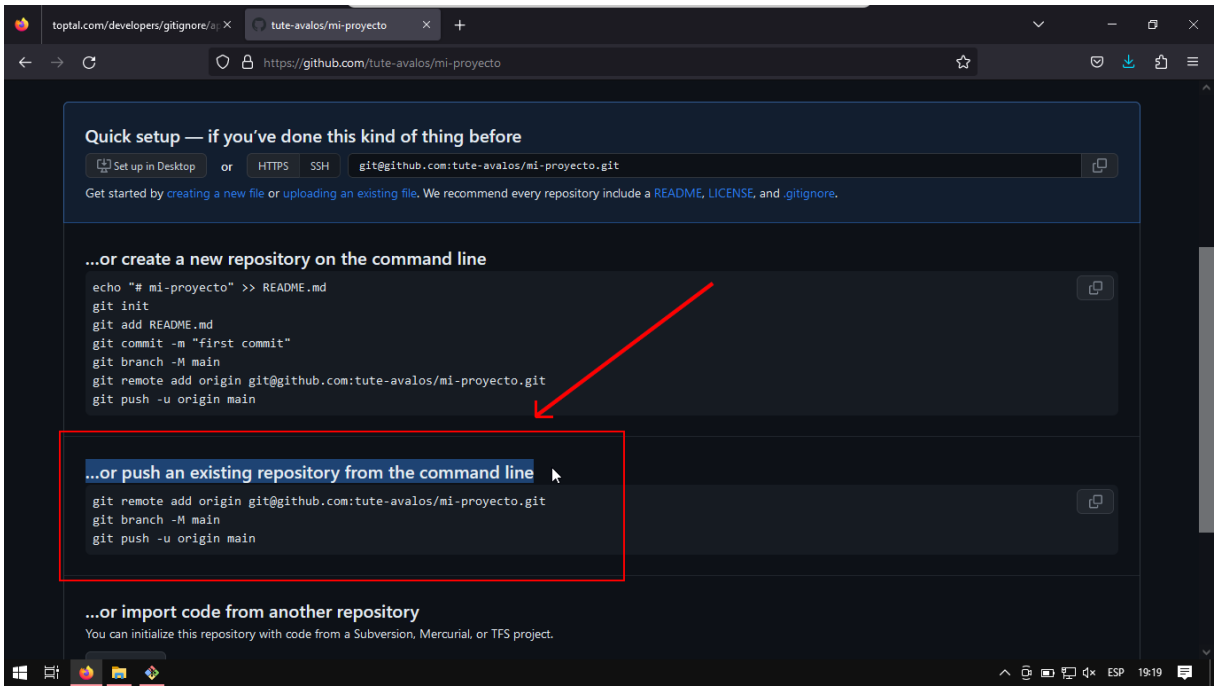


Finalmente, y **sin tildar nada más**, crearemos el repositorio remoto inicialmente vacío.



6. Enviar cambios a un repositorio remoto

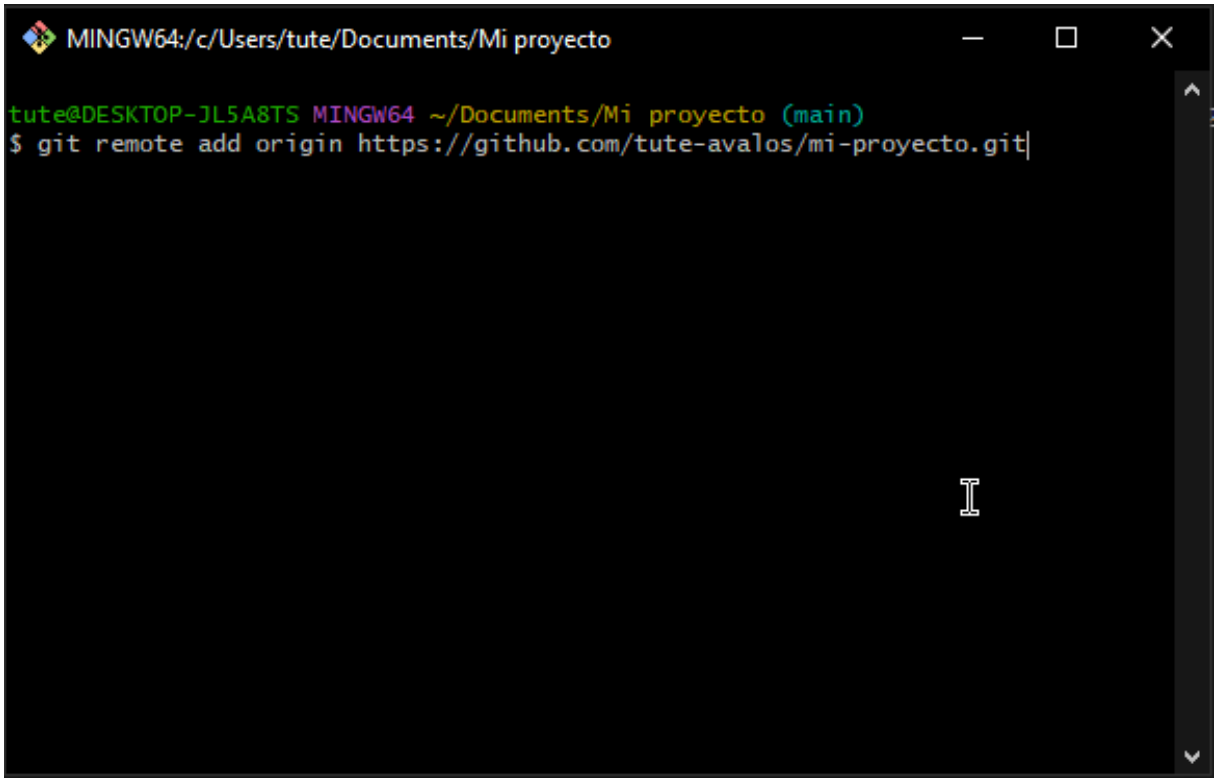
Una vez creado el repositorio remoto, debemos subir nuestro repositorio local ya existente. Esto puede lograrse fácilmente si seguimos las mismas instrucciones que nos da GitHub:



Aquí se está utilizando el sub-comando **`remote`** el cual gestiona los repositorios remotos. En este caso se utiliza **`add`** con el cual indicaremos un nombre para esa rama remota y una URL:

```
$ git remote add <nombre> <url>
```

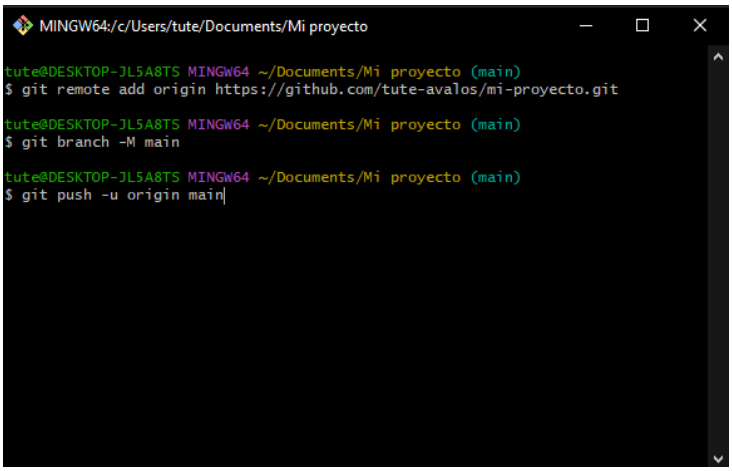
Para los fines prácticos, llamaremos **`origin`** a la rama remota indicando que es el *origen* del código, es decir que la rama remota será "la fuente del código original". Esta es la rama remota por omisión, es decir que si no se indica otra cosa, siempre será a esta rama.



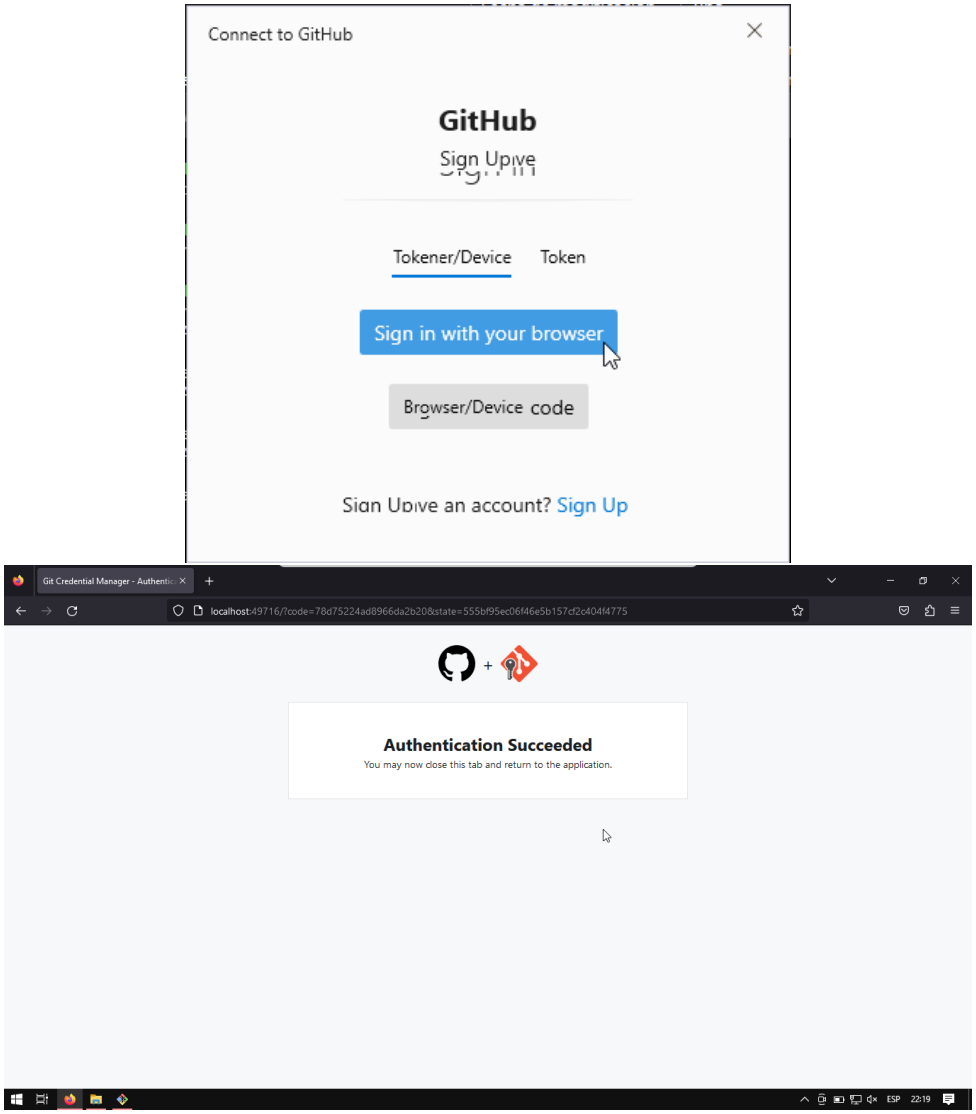
Finalmente se utilizará el sub-comando **push** para enviar todos los cambios (commits) del repositorio local al servidor:

```
$ git push -u <nombre> <rama>
```

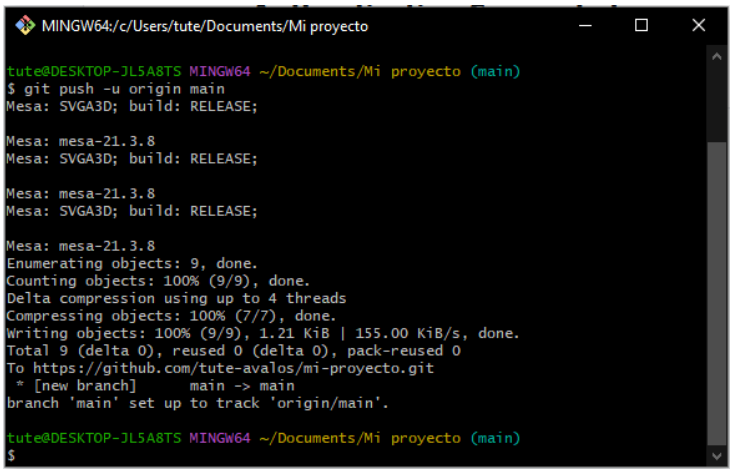
En este el parámetro **-u** indicará que queremos hacer el seguimiento de la rama remota, es decir que git esté pendiente si hubo cambios en el servidor o nuestra rama local sea distinta a la remota.

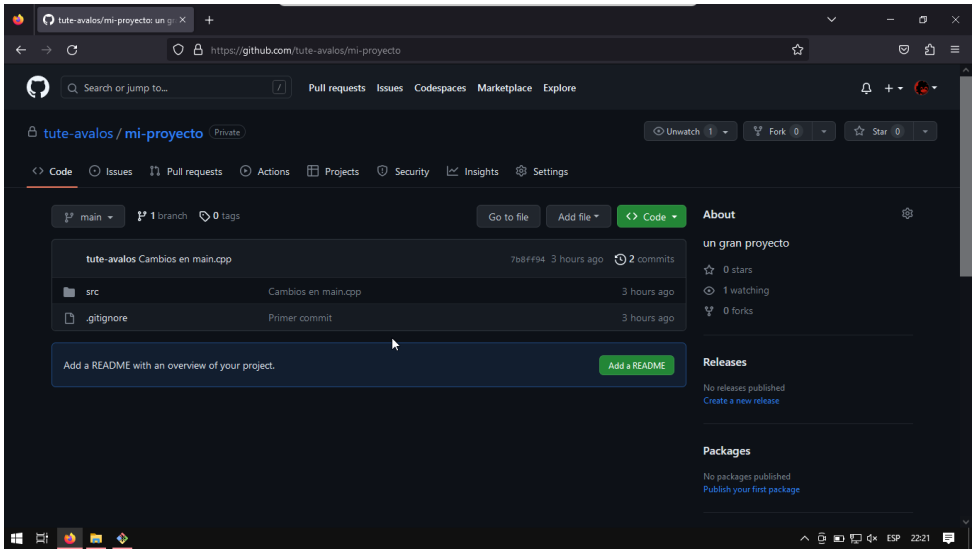


En este caso debemos autenticar nuestra cuenta en **Git Bash** para poder tener acceso con nuestra clave de GitHub:



Finalizado este proceso, podremos observar que todos los cambios fueron realizados en el servidor.





Así quedaría el nuevo flujo de trabajo si agregamos este último paso: