



Software de Control de Versiones

Introducción

Git es una herramienta de desarrollo de software que permite el control de versiones de un proyecto cualquiera, ya sea de código de programación u otro.

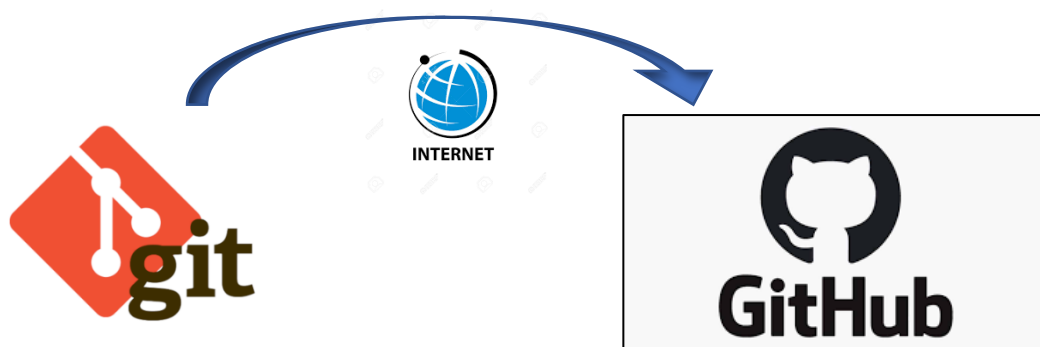
Guarda un snapshot de cada momento del proyecto. Almacena la primera versión de todas del proyecto y, para las posteriores, sólo guarda los registros de todos los cambios que se fueron realizando.

En otras palabras, permite guardar todo el historial completo de todo el proyecto que se haya desarrollado.

Cada vez que se guarda una nueva versión, se puede mover entre las versiones anteriores (*rollback*) y posteriores del proyecto.

Almacenado en línea

Github (y también GitLab) permite el almacenado de versiones en internet, permitiendo así el trabajo cooperativo.



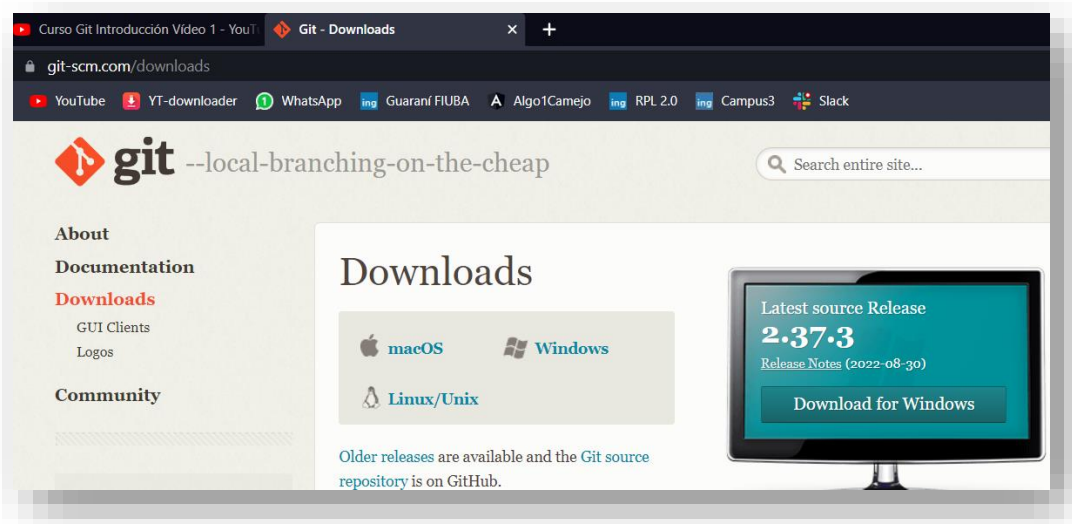
Repositorio

Carpeta con los archivos de nuestro proyecto y unos archivos ocultos que permiten la funcionalidad del programa:

- **gitignore** Lista de todas las extensiones de archivos que no se almacenarán dentro de cada backup.
- **git** Archivo interno que permite el control de versiones, es decir, almacena el orden de las versiones y las ramas del proyecto.

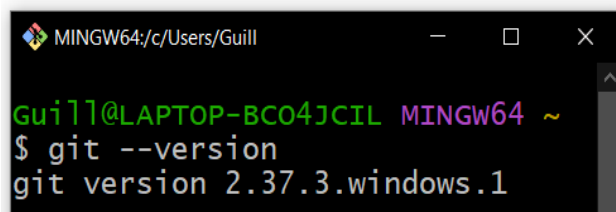
Guía de instalación

1. Crear usuario en GitHub para luego poder usar ese mismo usuario en Git.
2. Descargar GIT desde su página: <https://git-scm.com/downloads>



En caso de Windows, se debe instalar también github bash (emulador de terminal)

Una vez instalado, ejecutar la terminal (github bash en Windows) y ejecutar el comando "git --version" para corroborar que se haya instalado correctamente.



A continuación, en la misma terminal, se puede configurar nuestra cuenta de Git mediante las siguientes líneas de comando:

```
git config --global user.name "Guillermo Hernandez"
git config --global user.email ghernandez@fi.uba.ar
git config --global core.editor "code --wait"
```



requiere VS Code

Luego, visualizar la configuración hecha con el comando:

```
git config --global -e
```

Se debería abrir el IDE de Visual Studio Code, tras cerrar la ventana, quedará establecido como nuestro editor de texto por defecto para Git.

Además, según el sistema operativo utilizado, se debe realizar la siguiente configuración:

```
git config --global core.autocrlf true → WINDOWS
git config --global core.autocrlf input → LINUX
```

Para poder ver todas las configuraciones disponibles:

```
git config -h
```

Comandos básicos de la terminal

- `pwd` Muestra el directorio / carpeta actual.
- `ls` Muestra los archivos del directorio. `ls -a` Muestra incluso los ocultos
- `cd` Permite ingresar en un directorio (`..` permite ir hacia atrás)
- `mkdir` Permite crear un nuevo directorio.
- `rm` Permite eliminar un archivo.
- `code .` Permite abrir el IDE configurado por defecto.

Creación de un nuevo proyecto

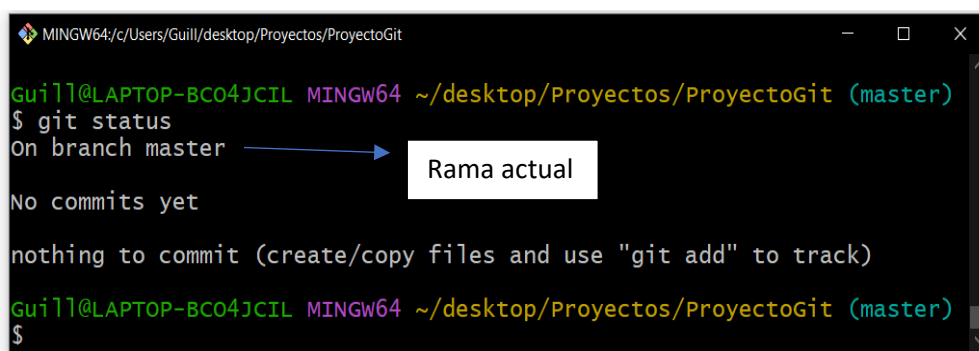
Para comenzar un nuevo proyecto, es necesario ubicarnos, con la terminal, en el directorio donde se ubicará el proyecto. Una vez ubicados allí, se procede a utilizar el comando `git init`. De esta manera, dicho directorio será convertido en un **repositorio**, creando el archivo `.git`

Para crear el archivo `.gitignore`, el cual permite que se ignoren determinadas extensiones, se crea el archivo y se enlistan todas las extensiones no deseadas. A continuación se guarda y se ejecuta: `git commit -a .gitignore`. Una página útil puede ser <https://www.toptal.com/developers/gitignore>

Áreas y/o etapas

WORKSPACE	STAGE	COMMIT	internet	SERVER
Donde se encuentra nuestro proyecto localmente alojado y visible desde el IDE utilizado (VS Code).	Archivos seleccionados que están esperando para poder ser añadidos a una nueva versión del proyecto, alojado en Git .	Archivos que ya componen una versión del proyecto, alojado y guardado en Git .		Repositorio remoto.

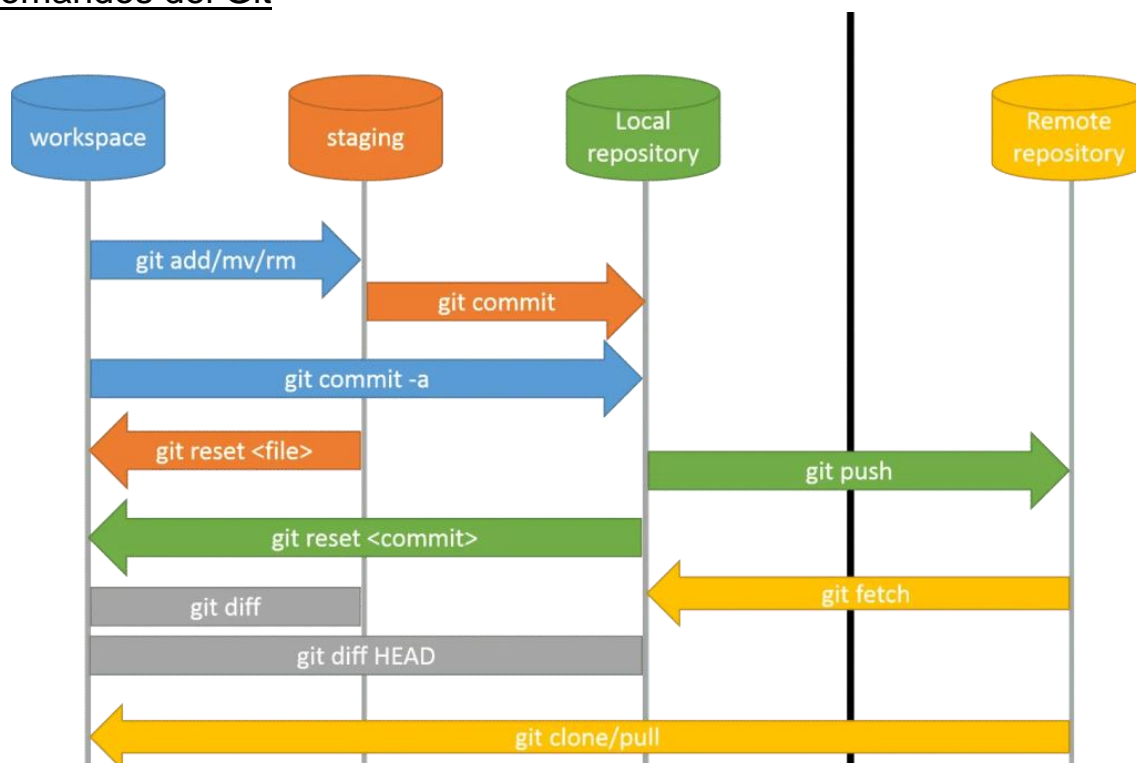
Mediante el comando `git status` se puede verificar en qué estado se encuentra el proyecto. Si se agrega el parámetro `-s`, se puede visualizar de una forma resumida.



```

MINGW64/c/Users/Guill/desktop/Proyectos/ProyectoGit
Guill@LAPTOP-BC04JCIL MINGW64 ~/desktop/Proyectos/ProyectoGit (master)
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
Guill@LAPTOP-BC04JCIL MINGW64 ~/desktop/Proyectos/ProyectoGit (master)
$
  
```

Comandos del Git



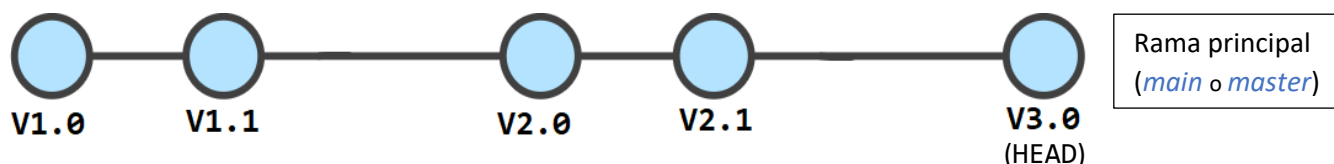
- ❖ **`git add archivo.xxx`** Añade (copia) un archivo al Stage Area de Git.
 Se puede utilizar **`*.xxx`** para agregar todos los de ese tipo.
 También se puede utilizar **`.`** para añadir todos los archivos.
 Es necesario aclarar que el archivo movido (copiado) al Stage Area ya no sufrirá cambios. Todas las ediciones en el código se harán en el Workspace, y para actualizar el archivo del Stage Area es necesario volver a ejecutar el comando. Para quitarlo: “`git rm –cached archivo.xxx`”
- ❖ **`git commit -m ""`** Compromete los archivos del Stage Area para que formen parte de una nueva versión del proyecto. Se debe colocar un mensaje breve y descriptivo de la nueva versión entre las comillas.
- ❖ **`git rm archivo.xxx`** Elimina un archivo y lo actualiza en el Stage Area. Es necesario ejecutar el *commit* para actualizarlo en la nueva versión. Para recuperar el archivo eliminado, se utiliza **`git restore archivo.xxx`**
- ❖ **`git mv archivo.xxx nuevo_nombre.xxx`** Modifica el nombre de un archivo y lo actualiza en el Stage Area. Es necesario ejecutar el *commit*.



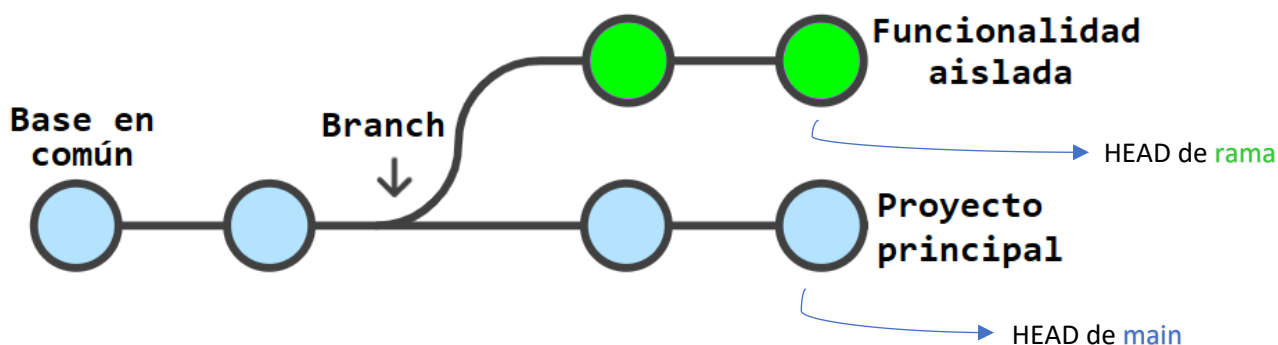
- ❖ `git dff` Permite visualizar los cambios realizados con la anterior versión.
Si se utiliza `--staged` muestra los cambios con los del Stage Area.
- ❖ `git log --oneline` Visualiza los cambios realizados por cualquier usuario, indicando la versión y la descripción brindada.
- ❖ `git revert HEAD` Permite volver a una versión anterior, sin borrar nada. Una vez que se actualize una nueva versión con *commit*,
- ❖ `cat archivo.xxx` Visualiza el contenido de un archivo.
- ❖ `git status -s` Permite visualizar el estado de todos los archivos y la rama actual en la que se encuentra trabajando de forma resumida.
 - Color
 - **Rojo** archivo en Workspace o no actualizado.
 - **Verde** archivo actualizado en Stage Area.
 - Letra
 - **A** añadido
 - **M** modificado
 - **D** eliminado
 - **???** no trackeado (nunca estuvo en Stage Area)

Branch (rama)

El historial de versiones de Git se realiza sobre un tronco o rama principal llamada *main* o *master*. Cada vez que se realiza un *commit*, es decir, que se crea una nueva versión del proyecto, dicha rama se “alarga”. La última versión encontrada de una rama se denomina **head**.



Sin embargo, es posible aislarse de la rama principal para trabajar en el proyecto de forma separada al proyecto principal, por ejemplo, para crear una funcionalidad específica sin necesidad de estar modificando el proyecto completo. Esto es especialmente útil cuando se trabaja de forma cooperativa.



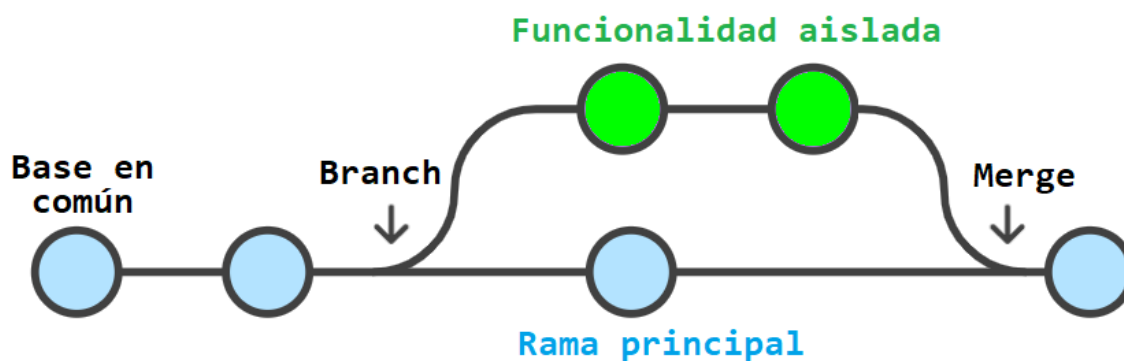
Para conocer en qué rama se está trabajando actualmente, se puede utilizar el comando `git branch`.

Para crear una nueva rama, se utiliza el comando `git checkout -b` seguido del nombre de la rama. En empresas se escribe *features/nombre-funcionalidad* o incluso un código. De esta forma, automáticamente nos situará en la rama creada.

Para pasar de una rama a otra, se utiliza `git checkout nombre-de-la-rama`

Merge

La función de crear ramas (branch) en un proyecto es la de trabajar de forma separada dentro de un mismo proyecto para, en algún momento, unir las ramas de nuevo a la principal. Este proceso se llama *merge*.



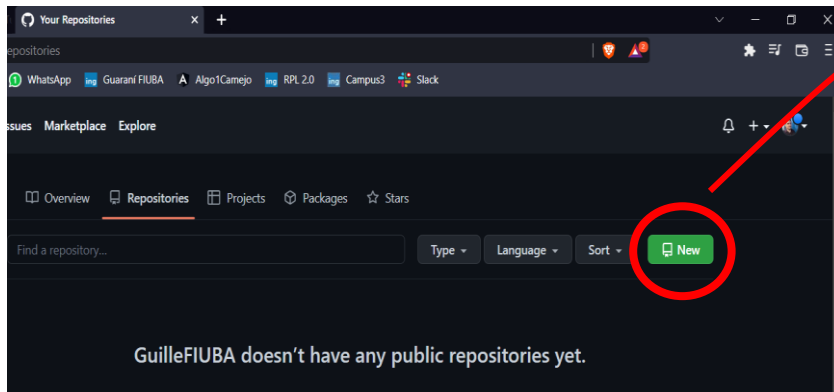
Para realizar un merge, **es necesario estar ubicado en el HEAD de la rama de recepción**, es decir, la que recibirá el merge. En este caso, sobre el HEAD de la rama principal.

Una vez situado sobre el HEAD receptor, se ejecuta:

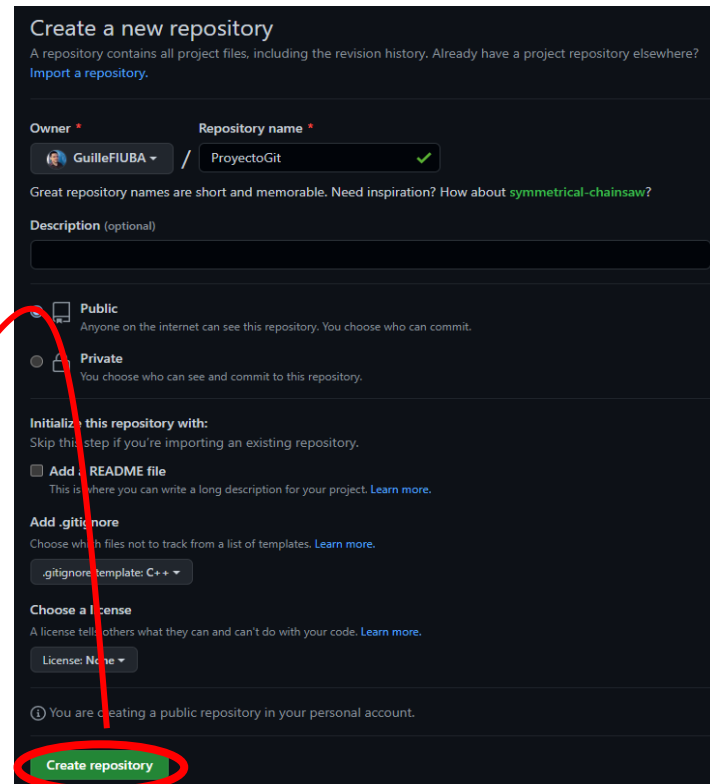
```
git merge nombre-de-la-rama-que-se-unirá
```

Sincronización del proyecto

Para realizar un proyecto cooperativo con un servidor, se puede utilizar GitHub.



Crear un repositorio remoto



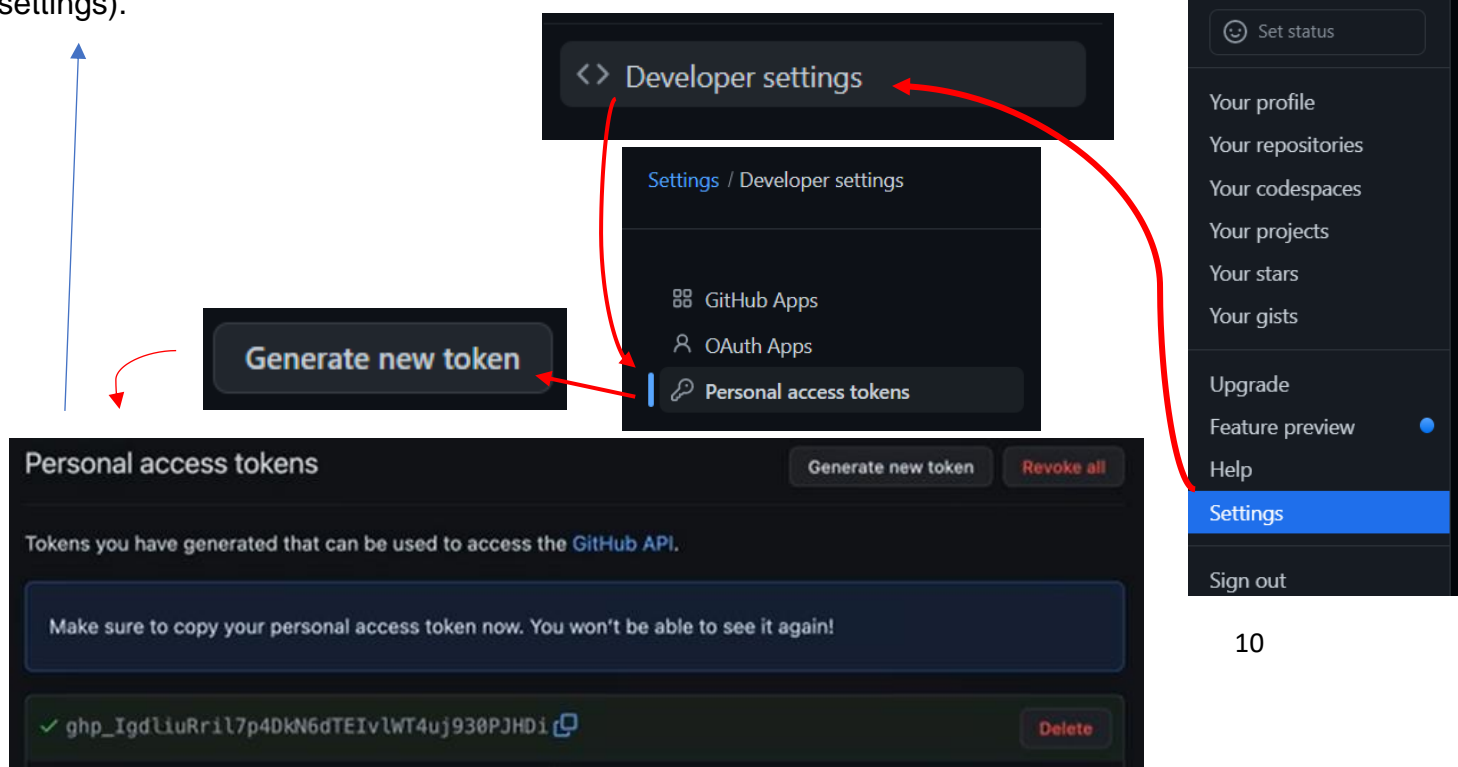
Una vez creado el repositorio remoto en GitHub, se procede a ejecutar las siguientes líneas de comandos que nos brinda GitHub para sincronizarlo con nuestro Git.

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

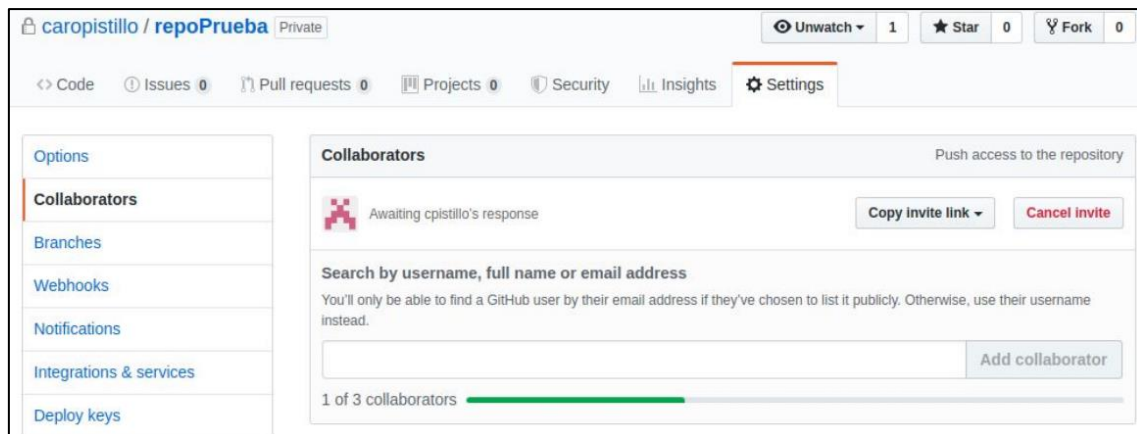
echo "# miweb" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/nschurmann/miweb.git
git push -u origin main
```

El parámetro **-u** indica que se creará una rama, **origin** indica que dicha rama será la principal, y **main** será el nombre de la rama principal.

Tras hacer esto, se nos pedirá un nombre de usuario y una contraseña (la cual no es la de nuestra cuenta de GitHub, sino el token que se puede generar desde settings).



Para invitar a un colaborador, se hace en la siguiente ventana:



Iniciar un proyecto directamente desde GitHub

Otra forma de iniciar un proyecto, es desde GitHub.

Una vez creado el proyecto, se copia el link del repositorio remoto de GitHub y se utiliza `git clone LINK` en la terminal, estando ubicados en la carpeta donde estará alojado nuestro repositorio local. A continuación, en el mismo directorio, se debe ejecutar `touch .gitignore` para crear dicho archivo.

Workflow genérico

1. `git pull` Trae la versión más reciente del repositorio de GitHub.
2. Hacer los cambios deseados y luego “commitear”: `git add` y `git commit -m "..."`
3. `git pull` Trae la versión más reciente del repositorio de GitHub y lo fusiona con el WorkSpace, necesario para comprobar que no haya conflictos.

NUNCA HACERLO SIN HABER COMITEADO PRIMERO

4. Arreglar conflictos del merge en caso de ser necesario y *commitear*.
5. `git push` Crea una nueva versión en el repositorio de GitHub con el contenido del repositorio local.

Workflow con más ramas

Para crear una nueva rama en GitHub, es necesario utilizar:

```
git push --set-upstream origin nuevaBranch
```

Esta forma de “pushear” es sólo necesaria para la creación de la rama, luego se puede realizar de forma normal.

1. `git pull` Trae la versión más reciente del repositorio de GitHub.
2. Hacer los cambios deseados y luego “commitear”: `git add` y `git commit -m "..."`
3. Combinar la nueva rama con la original o rebase a la rama original

Combinar con la rama original	Rebase a la rama original
<pre>git checkout ramaOriginal</pre> <pre>git merge nuevaRama</pre> <p>Hacer <i>commit</i> luego de arreglar los conflictos</p>	<pre>git rebase --continue</pre>

4. `git push` Crea una nueva versión en el repositorio de GitHub con el contenido del repositorio local.