

Git y Github

¿Qué es un control de versiones?

- Es un sistema que registra cada cambio que se realiza en el código fuente de un proyecto.
- Permite tener un historial de todos los cambios, saber quién lo hizo y cuándo, crear copias de seguridad de nuestro código, y recuperar versiones específicas del proyecto.
- También permite que diferentes personas puedan colaborar en el desarrollo de una misma aplicación, facilitando la revisión, sincronización y versionado de los cambios.

¿Qué es Git?

Git es un sistema distribuido de control de versiones, gratuito y de código abierto bajo licencia GPLv2.

Los repositorios

Un repositorio es una carpeta en la que se almacenan las diferentes versiones de los ficheros de un proyecto y el histórico de los cambios que se han realizado en ellos.

Tipos de Sistema de Control de Versiones

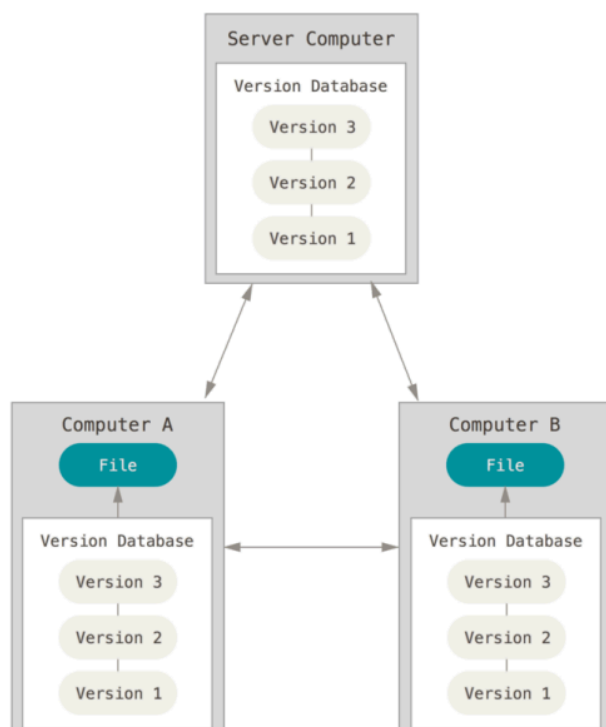
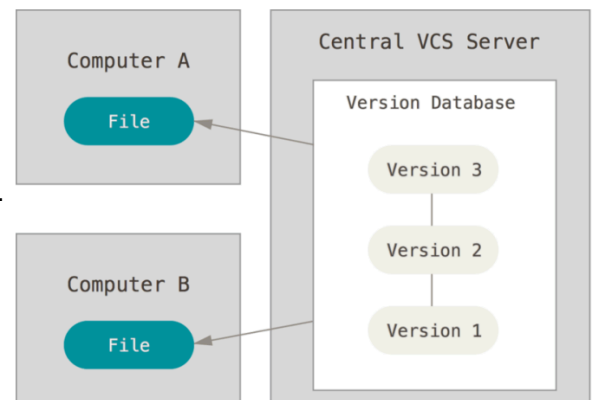
LOCALES: Los tenemos en nuestro ordenador. Son los menos útiles.

REMOTOS: Ubicados en un servidor externo. Son los que permitirán que otras personas puedan trabajar en el mismo proyecto, hacer cambios y que puedan ser sincronizados, y permiten que los administradores tengan control sobre qué puede hacer cada usuario.

Los Sistemas de Control de Versiones (SCV) Remotos pueden ser Centralizados o Distribuidos:

SCV Remotos Centralizados: Ubicados en un servidor externo.

La desventaja es que si el servidor centralizado se cae no se puede trabajar, y si se corrompe el disco duro donde está la base central se pierde todo.



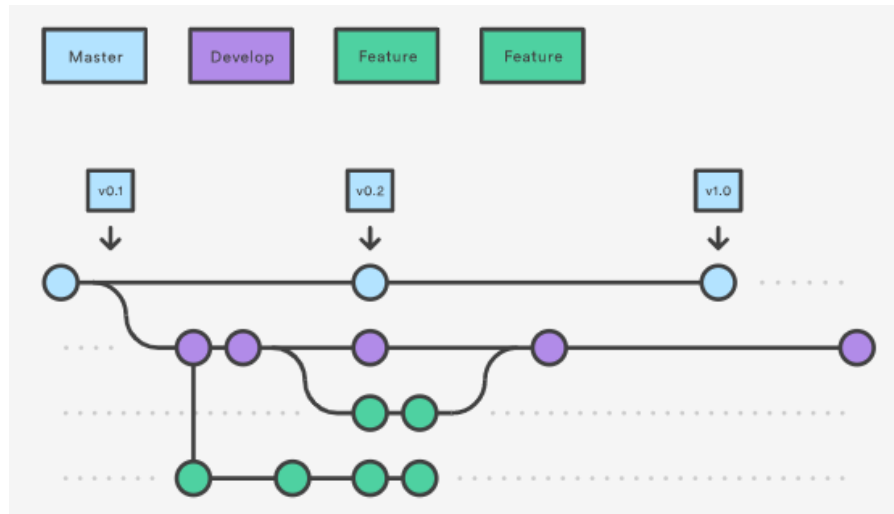
SCV Remotos Distribuidos: ES EL CASO DE GITHUB.

Los clientes no solo descargan la última copia de los archivos, sino que se clona completamente el repositorio con todos los datos. Así, si un servidor deja de funcionar, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo.

Master y Ramas

Cada repositorio tiene, al menos, una rama principal llamada master o main, y las ramas que se crean en ella serían las ramas de desarrollo o branches.

Esta es sólo una estrategia de las muchas que existen a la hora de trabajar con Git y similares.



Una rama es simplemente una versión de la colección de directorios y archivos del repositorio. Cada vez que se crea una nueva rama, se crea una copia de la colección de archivos actual.

Al hacer cambios, éstos se confirman y se guardan (en el master o en la rama, donde se esté trabajando).

En un entorno de colaboración, donde diferentes personas están trabajando en un mismo código, se genera una evolución del código en paralelo: mientras alguien está trabajando en añadir una nueva característica al proyecto, otra persona puede estar arreglando un bug y otra en añadir alguna documentación.

Merge, Push, Pull

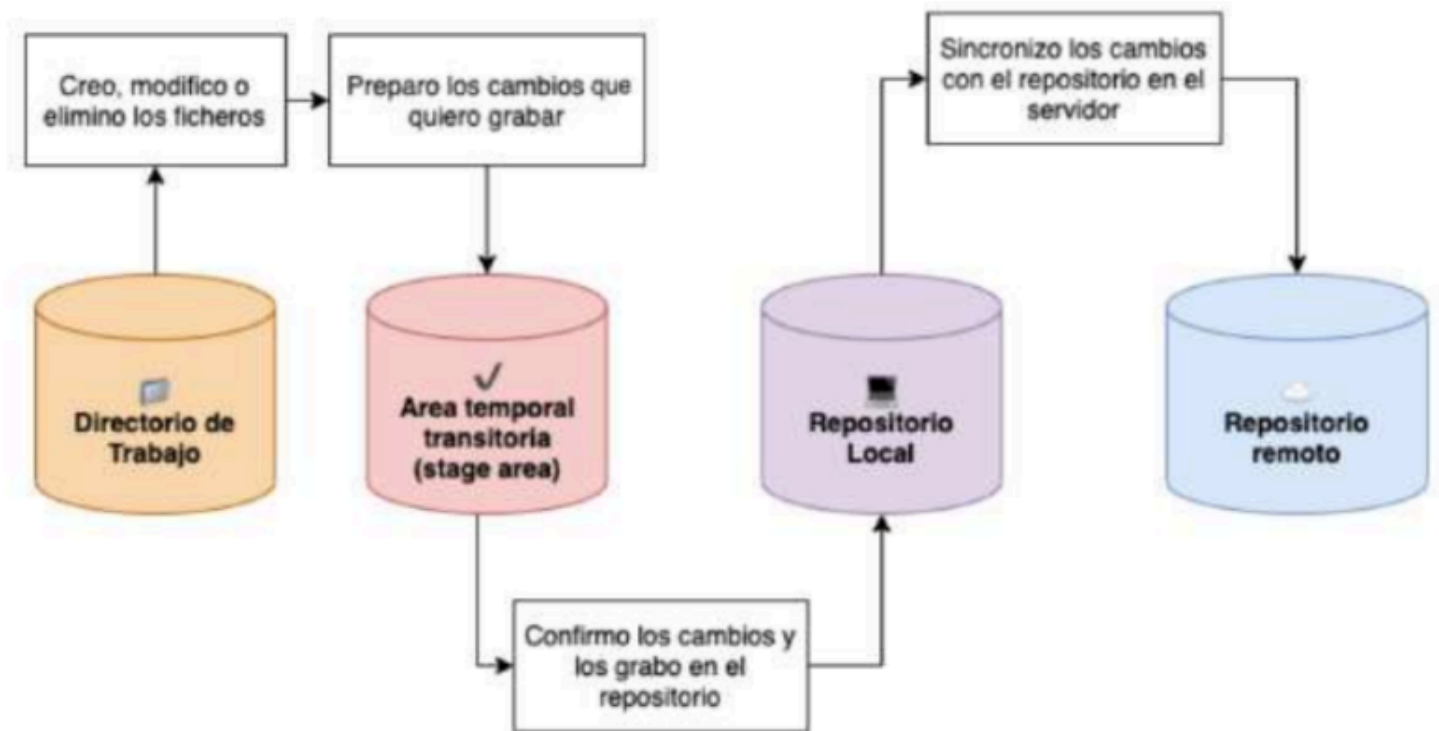
Merge: Proceso que consiste en combinar los cambios de una bifurcación con otra rama o con la rama principal.

Push: Consiste en llevar (o empujar) los cambios locales al repositorio remoto

Pull: Es la acción inversa, traer los cambios del repositorio remoto al local.

Los 3 estados de los archivos del proyecto GIT

- **Modificado (modified):** El archivo contiene cambios, pero todavía no han sido marcados para ser confirmados. Se encuentra en el directorio de trabajo.
- **Preparado (staged):** Son los archivos que han sido modificados en el directorio de trabajo y se han marcado como preparados para ser confirmados en el repositorio local. Se encuentran en un área temporal transitoria (staging index). Esta acción recibe el nombre de add.
- **Confirmado (committed):** El archivo se encuentra grabado en el repositorio local. Esta acción recibe el nombre de commit.



¿Qué es una consola / terminal de comandos?

Es una interfaz de texto que permite a los usuarios ingresar comandos y realizar tareas en un sistema operativo.

Se utiliza para interactuar con el sistema operativo y realizar tareas de manera más eficiente que a través de la interfaz gráfica de usuario.

Están disponibles en la mayoría de los sistemas operativos, incluyendo Windows, MacOS y Unix/Linux.

¿Por qué una consola es más eficiente que un GUI?

- **Velocidad:** Las consolas son más rápidas que las interfaces gráficas ya que no requieren tiempo para dibujar elementos en la pantalla.
- **Automatización:** Las consolas permiten la automatización de tareas mediante el uso de scripts y programas de línea de comando, lo que ahorra tiempo y esfuerzo.
- **Acceso a funciones avanzadas:** Las consolas ofrecen acceso a funciones avanzadas que no están disponibles en la interfaz gráfica, lo que las hace más flexibles y potentes.
- **Control preciso:** Las consolas permiten un control más preciso del sistema operativo, lo que es útil para solucionar problemas y realizar tareas de mantenimiento.

Operaciones básicas con archivos (bash command)

Crear directorio: md "Nombre del directorio"

"md" significa "Make Directory" (Crear Directorio). Seguido de ese comando debe ingresarse el nombre del directorio que queremos crear.

```
C:\Users\dany\Desktop> md "Diplomatura de Java"
```

Cambiar/Moveirse de directorio: cd "nombre del directorio"

"cd" significa "Change Directory" (Cambiar Directorio). Seguido de ese comando debe ingresarse el nombre del directorio al que queremos movernos.

```
C:\Users\dany\Desktop> cd "Diplomatura de Java"  
C:\Users\dany\Desktop\Diplomatura de Java>
```

Regresar al directorio padre: cd ..

Este comando te regresará al directorio padre del directorio actual.

```
C:\Users\dany\Desktop\Diplomatura de Java> cd ..  
C:\Users\dany\Desktop>
```

Listar el contenido de un directorio: dir

Podremos visualizar todos los archivos y directorios que se encuentran dentro del directorio actual.

Crear una copia de un archivo: copy "archivo" "directorio/archivo copia"

Este comando creará una copia de un archivo en la ruta y con el nombre que le indiquemos. El primer dato es el nombre del archivo a copiar, y el segundo dato es la ruta junto con el nombre de la copia de ese archivo que se va a crear.

```
C:\Users\dany\Desktop\Diplomatura de Java>copy "archivo 1.txt" "Archivo 1 copia.txt"  
1 archivo(s) copiado(s).
```

Mover archivo: move "archivo" "directorio\archivo"

Este comando mueve un archivo de un directorio a otro, el primer dato es el nombre del archivo y el segundo dato es el directorio donde se va a mover junto con su nombre (opcional).

```
C:\Users\dany\Desktop\Diplomatura de Java> move "archivo 1.txt" "carpeta\archivo 1.txt"  
Se han movido 1 archivos.
```

Crear un archivo: echo "contenido" > "nombre_archivo.formato"

Este comando creará un archivo dentro del directorio donde es invocado. El primer dato es el contenido que tendrá el archivo a crear, y el segundo dato es el nombre del archivo con su respectivo formato.

```
C:\Users\dany\Desktop> echo Hola Mundo! > archivo.txt
```

Eliminar un archivo: del "nombre del archivo"

"del" proviene de "Delete" (Eliminar).

```
C:\Users\dany\Desktop> del archivo.txt
```

Ver el contenido de un archivo: type "archivo.formato"

Este comando muestra el contenido del archivo indicado en la terminal.

```
C:\Users\dany\Desktop> type archivo.txt
Hola Mundo!
```

Estos son solo algunos de los comandos más utilizados. Hay muchos otros comandos disponibles en la línea de comandos de Windows que puedes explorar y aprender.

Limpiar la terminal: cls

"cls" proviene de "Clear System" (Limpiar Sistema).

¿Cómo configurar GIT por primera vez?

Paso 1: Elegir un nombre de usuario y el email que usarás en Github. Esto es importante porque las confirmaciones de cambios (commits) en Git usan esta información, y es introducida de manera inmutable en los commits que envías. El comando git config se usa para configurar el usuario.

Paso 2: Establecer el nombre con el comando: git config --global user.name "Nombre Apellido".

Paso 3: Establecer el correo a usar con el comando. git config --global user.email Mail@example.com.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Crear un primer proyecto

Imaginemos que una carpeta contiene un proyecto que necesitamos compartir mediante Github.

Como primer paso: Abramos la carpeta.

Una vez dentro de la carpeta del proyecto, vamos a darle click derecho para que nos aparezcan las siguientes opciones: Git Bash y Git GUI.

Git GUI: Nos proporciona una interfaz gráfica para trabajar, pero está algo desactualizada.

Git Bash: Abre una consola de comandos dedicada a Git.

Se nos debería abrir una ventana como ésta, en la cual ya vamos a proceder a escribir los comandos necesarios para lo que buscamos:

```
MINGW64:/c:/Users/Nahuel/Desktop/java-git-prueba
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba
$
```

Al momento de ejecutar “git init” en nuestra terminal se crea un repositorio local basado en la carpeta donde estamos parados.

Nótese la nueva notación “(master)” ... La palabra “master” hace referencia a que estamos situados justo en la carpeta raíz o la carpeta padre de todo nuestro repositorio, dándole a ésta el rango/categoría más importante posible.

```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba
$ git init
Initialized empty Git repository in C:/Users/Nahuel/Desktop/java-git-prueba/.git/

Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (master)
$
```

Una vez iniciado el repositorio, ya podemos proceder a subirlo a internet.

Hay que ver este proceso como si fuese el de enviar un paquete por correo.

Al ejecutar “git add .”, lo que estamos haciendo es añadir a una lista de “elementos pendientes por confirmar”, los archivos/cambios nuevos que hayamos hecho en nuestro código. En este caso como el repositorio es nuevo, pues estaríamos seleccionado todo el proyecto, para lo cual pondremos espacio y un punto.

```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (master)
$ git add .

Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (master)
$ |
```

Una vez que los ítems han sido añadidos a la lista de pendientes de confirmación, nos está faltando un paso importante: crear la rama en la cual serán subidos los cambios/archivos.

Para eso, con el comando “git branch -M main” lo que hacemos es crear una rama con el nombre de “main”, en la cual serán subidos todos los cambios de nuestro proyecto, al menos por ahora. Después podemos crear todas las que queramos.

Notese también que en vez de decir “máster”, ahora leemos “main”, ya que pasa a ser la rama principal del repo.

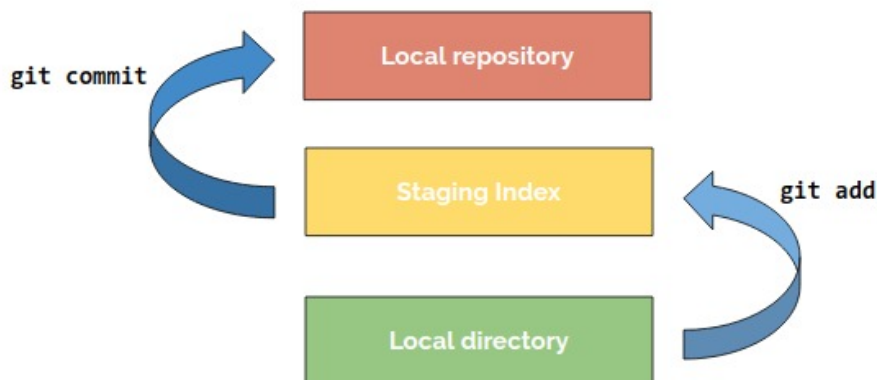
```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (master)
$ git branch -M main

Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (main)
$ |
```

Volviendo al ejemplo de enviar un paquete por correo, git add equivale a marcar los productos en una lista y fijar el destino. Ahora nos queda preparar el paquete...
La instrucción "git commit" equivale a preparar una caja llena de cambios lista para ser enviada e informar qué contiene ésta.

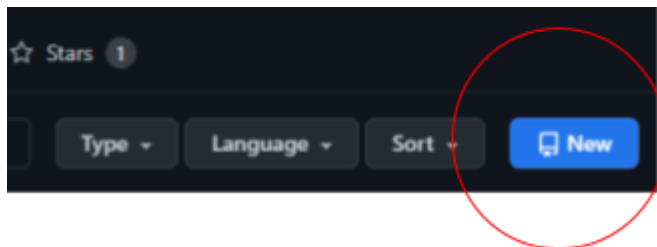
```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (main)
$ git commit -m "Este es mi primer commit"
[main (root-commit) d42dbe7] Este es mi primer commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644Codigo ;).txt
```

Workflow



Ahora vamos a GitHub!

Vamos a crear un repositorio desde acá, ya que necesitamos de un repositorio remoto para poder subir el código a internet.



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 * **NOMBRE**
java-git-prueba is available.

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

Description (optional) **DESCRIPCION**

☒ **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.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: None

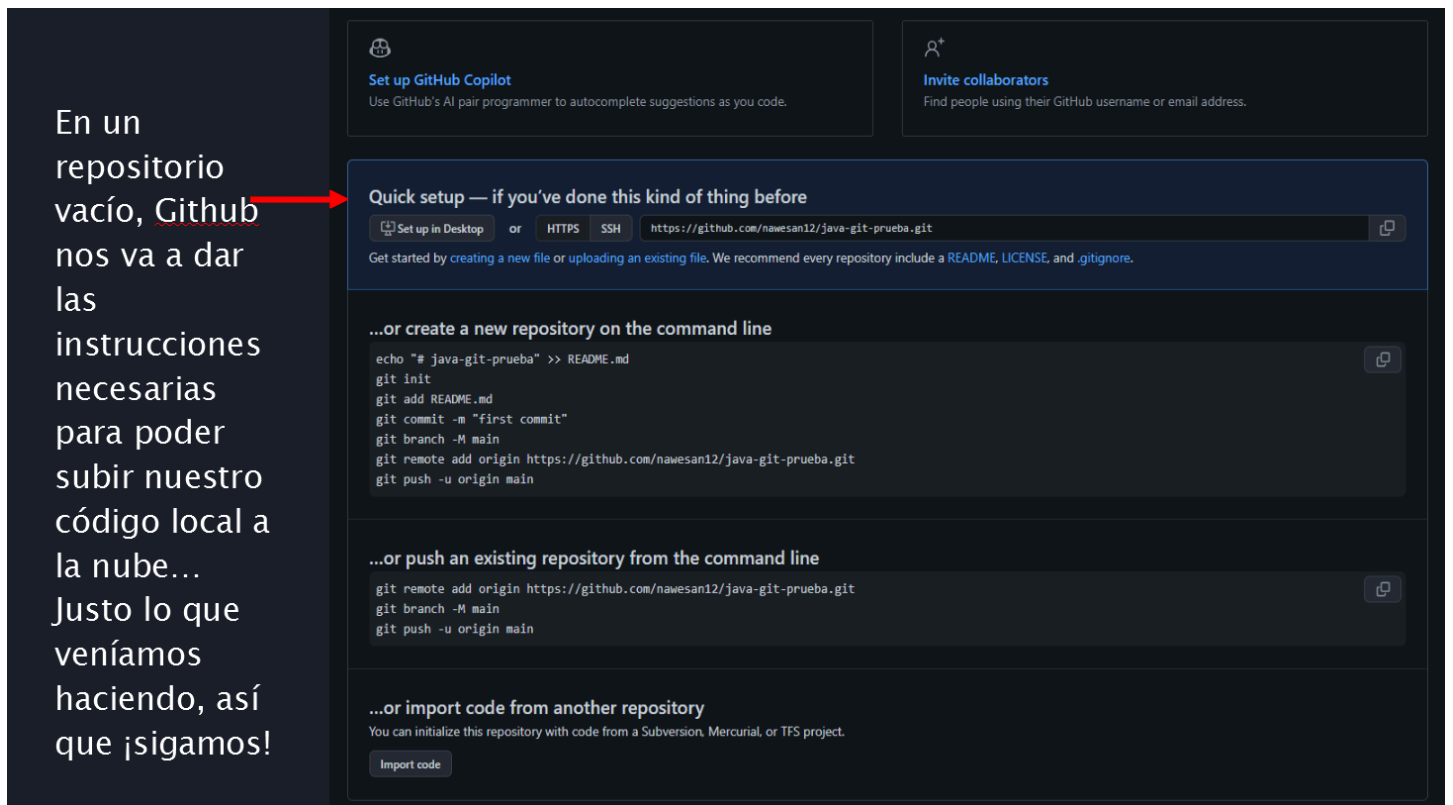
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

① You are creating a public repository in your personal account.

[Create repository](#)

En un repositorio vacío, Github nos va a dar las instrucciones necesarias para poder subir nuestro código local a la nube... Justo lo que veníamos haciendo, así que ¡sigamos!



Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Invite collaborators
Find people using their GitHub username or email address.

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

Set up in Desktop or HTTPS SSH `https://github.com/nawesan12/java-git-prueba.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 "# java-git-prueba" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/nawesan12/java-git-prueba.git
git push -u origin main
```

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

```
git remote add origin https://github.com/nawesan12/java-git-prueba.git
git branch -M main
git push -u origin main
```

...or import code from another repository

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

Import code

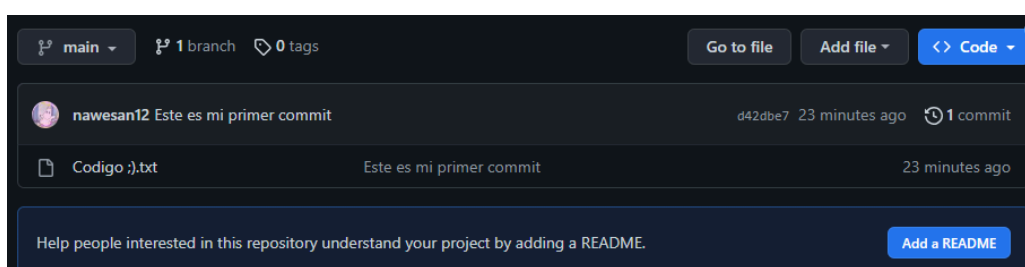
Ahora debemos declarar un origen remoto para el repositorio usando el comando “git remote add origin” y copiando a continuación el link del repositorio de Github en la web pero con la terminación “.git”. De esa manera reconocerá de qué repositorio estamos hablando y lo enlazará.

```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (main)
$ git remote add origin https://github.com/nawesan12/java-git-prueba.git
```

Por último... enviamos el paquete que veníamos preparando, usando el comando git push.

```
Nahuel@DESKTOP-GK4D1SJ MINGW64 ~/Desktop/java-git-prueba (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 227 bytes | 227.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/nawesan12/java-git-prueba.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Refrescamos la web de Github Y LISTO!



main 1 branch 0 tags

Go to file Add file <> Code

nawesan12 Este es mi primer commit d42dbe7 23 minutes ago 1 commit

Codigo :).txt Este es mi primer commit 23 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

Resumen

Para crear repo y subirlo por primera vez:

- `git init`
- `git add .`
- `git status`
- `git commit -m "Mi primer commit"`
- `git branch -m main`
- `git remote add origin "milink.git"`
- `git push -u origin`

Para subir solo cambios:

- `git add .`
- `git status`
- `git commit "Hola, soy un cambio"`
- `git push -u origin`

Para robar código clonar un repo:

- `git clone "link.git"`

Para traer cambios remotos de un repo que tengo desactualizado en el local:

- `git pull`

Para crear ramas:

- `git branch OPTIONS`