

# GIT & GITHUB



Francisco Valerio



# CONTENIDO

Control de versiones

Introducción a Git

Instalaciones

Básicos de Git

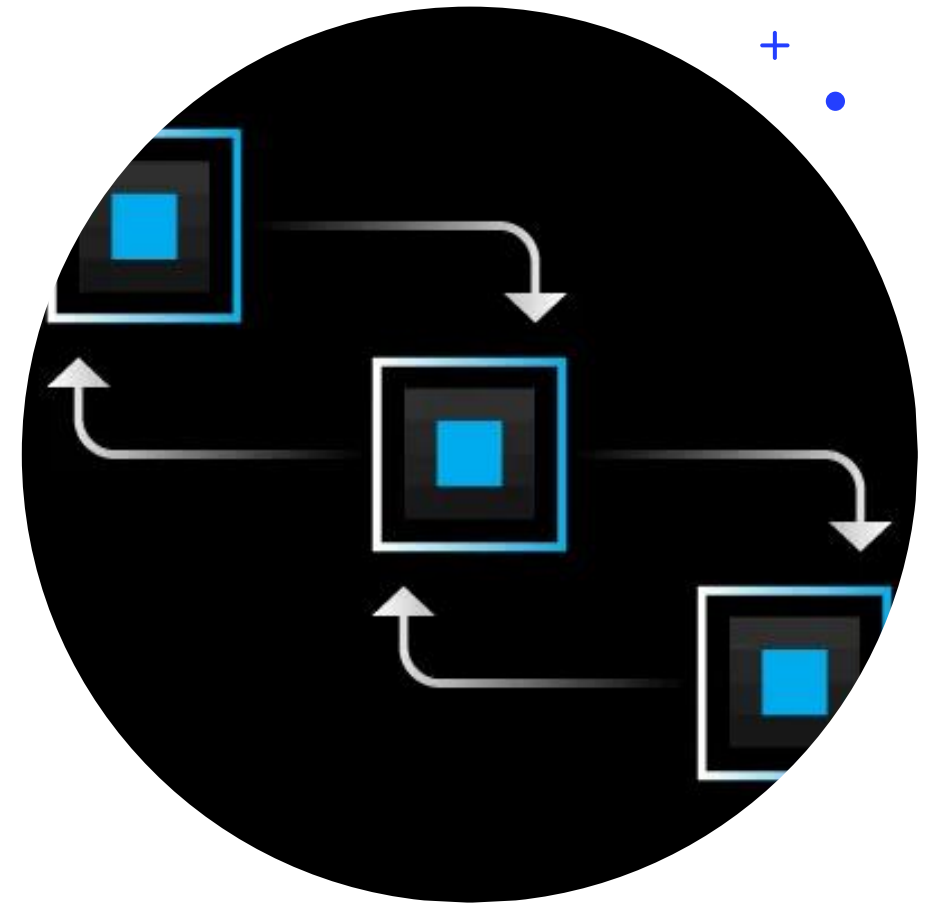
Introducción a Github

# Control de versiones

Un sistema de control de versiones (VCS) registra cambios a un archivo (o un conjunto de archivos) a lo largo del tiempo y que nos permite acceder a versiones anteriores específicas.

La principal ventaja es que podemos revertir los archivos que queramos a un estado previo, o incluso revertir un proyecto entero, comparar los cambios realizados y encontrar algún error.

En otras palabras, si arruinamos las cosas o perdemos archivos, los podemos recuperar fácilmente.





# INTRODUCCIÓN A GIT

# Git

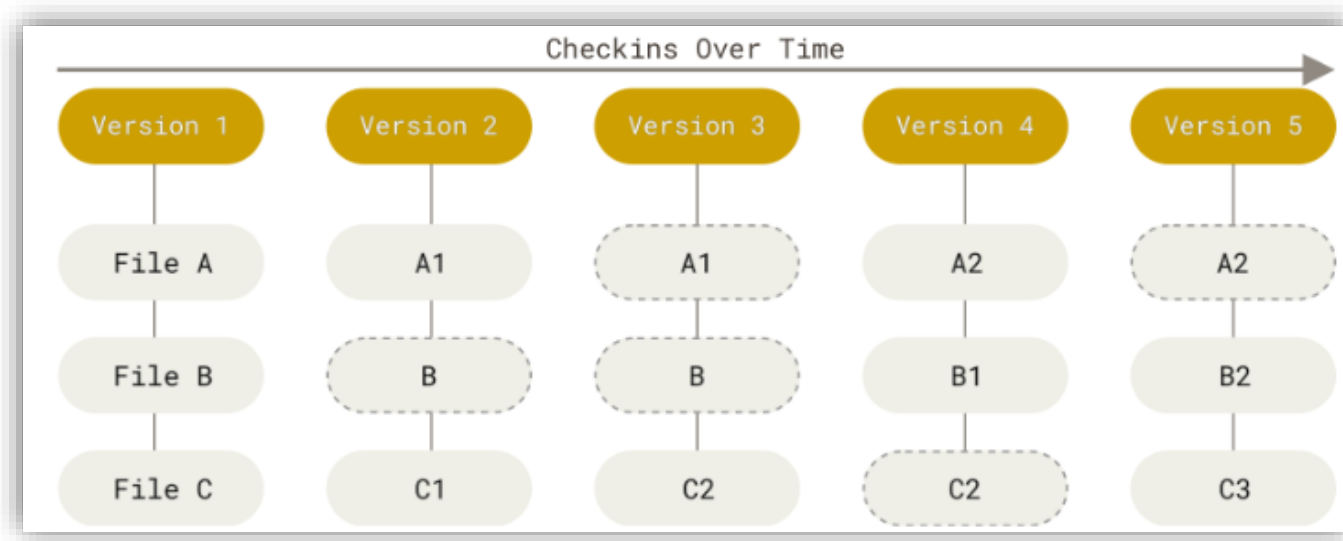


## ¿Qué es Git?

- Git es un VCS cuyo propósito es llevar el registro de los cambios realizados a archivos en una computadora, así como de coordinar el trabajo cooperativo de varias personas que modifiquen un mismo archivo.

## ¿Cómo funciona?

- Git considera a los datos como una serie de instantáneas de un sistema de archivos en miniatura.
- Cada que guardamos el estado de nuestro proyecto, Git “toma una foto” de cómo se ven los archivos en ese momento y almacena una referencia a esa instantánea.



# Git

## ¿Cómo funciona?

- Para mantener su eficiencia, si los archivos no han cambiado, no los vuelve a almacenar, sólo una referencia al archivo idéntico almacenado previamente.
- Git considera a los datos más como un flujo de instantáneas

## Integridad en Git

- Todo en Git es revisado antes de ser almacenado. Es imposible cambiar los contenidos de los archivos o carpetas sin que Git no lo sepa.
- No puedes perder información en tránsito o corromper tus archivos sin que Git no lo detecte.
- El mecanismo que utiliza Git para esta comprobación se llama hash SHA-1.
- Es una cadena de 40 caracteres hexadecimales y se calcula con base en los contenidos del archivo.

## Git opera localmente

- La mayoría de las operaciones en Git utilizan sólo archivos y recursos locales.
- El historial y cambios del proyecto pueden verse instantáneamente.

# Git

## Integridad en Git

Un hash SHA-1 se ve así:

A dark-themed terminal window with a light gray border. In the top right corner, there are three small icons: a minus sign, a square, and an 'X'. The main area of the terminal is black and contains a single line of white text representing a SHA-1 hash.

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Cuando utilicemos Git veremos estos valores porque Git almacena todo en su base de datos no por el nombre de archivo, sino por el valor hash de sus contenidos.

# Git

## Los tres estados

¡Atención! Aquí viene lo principal a recordar. Git tiene tres estados principales en los que tus archivos pueden estar: **modified**, **staged** y **committed**.

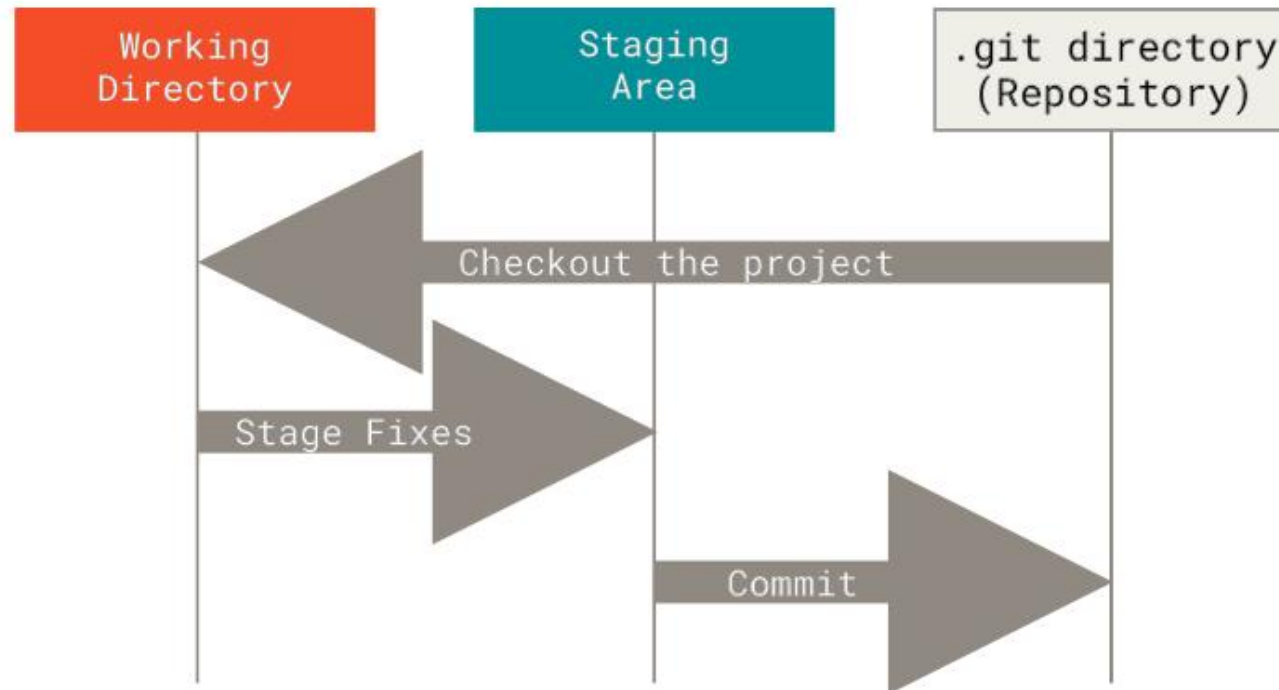
- **Modified:** significa que has cambiado el archivo pero que no lo has confirmado a tu base de datos aún.
- **Staged:** significa que has marcado un archivo modificado en su versión actual para confirmarlo en tu siguiente instantánea.
- **Committed:** significa que los datos se han almacenado de manera segura en tu base de datos local.



# Git

## Los tres estados

Esto nos lleva a las tres secciones principales de un proyecto de Git: el **working tree**, el **staging area**, y el **Git directory**.



# Git

## Flujo de trabajo

Entonces el flujo de trabajo básico en Git va así:

1. Modificamos los archivos en el working tree.
2. Seleccionamos sólo los cambios que queremos que sean parte de la siguiente etapa, lo que añade *sólo* esos cambios al staging area.
3. Hacemos un *commit* que toma los archivos como están en el staging area y almacena permanentemente esa instantánea al Git directory.



# INSTALACIONES

# Instalaciones

## Instalando Git

Para instalar Git basta con ir a la página de descarga. Actualmente está disponible para macOS, Windows y Linux/Unix.

[Git download](#)

Adicionalmente necesitamos de un editor de texto. Recomendando utilizar Sublime Text.

[Sublime Text download](#)



# Configuración inicial

## Setup inicial de Git

Una vez que instalamos Git en nuestro sistema, debemos personalizar algunas cosas en el ambiente de Git.

Git tiene una herramienta llamada `git config` que nos permite obtener y establecer variables de configuración que controlan los aspectos de cómo se ve y opera Git.

Para ello vamos a Inicio y buscamos “Git Bash” y ejecutamos la aplicación.

# Configuración inicial

## Setup inicial de Git

Cuando estemos en el Git Bash ejecutamos el siguiente comando para establecer nuestro nombre de usuario:

```
$ git config --global user.name "Francisco Valerio"
```

A continuación, establecemos nuestro correo electrónico:

```
$ git config --global user.email abcd@email.com
```

# Comandos básicos

## Versión de Git

```
$ git version
```

## Revisar la configuración

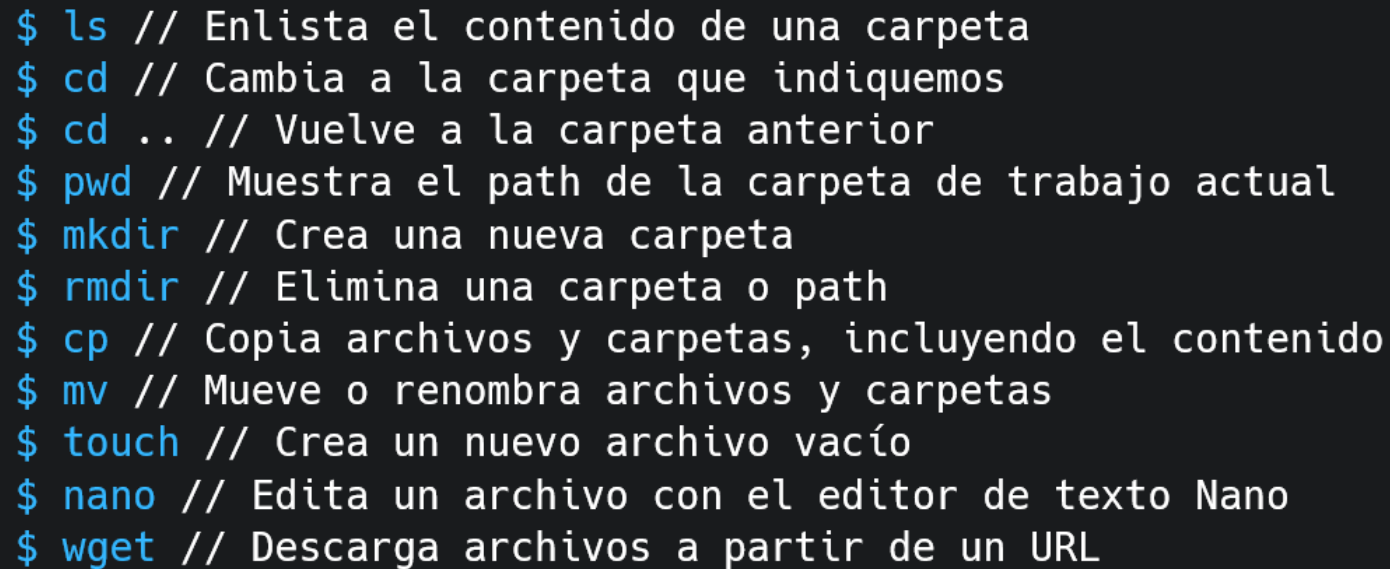
```
$ git config --list
```

## Pedir ayuda en Git

```
$ git help <verb>  
$ git <verb> --help
```

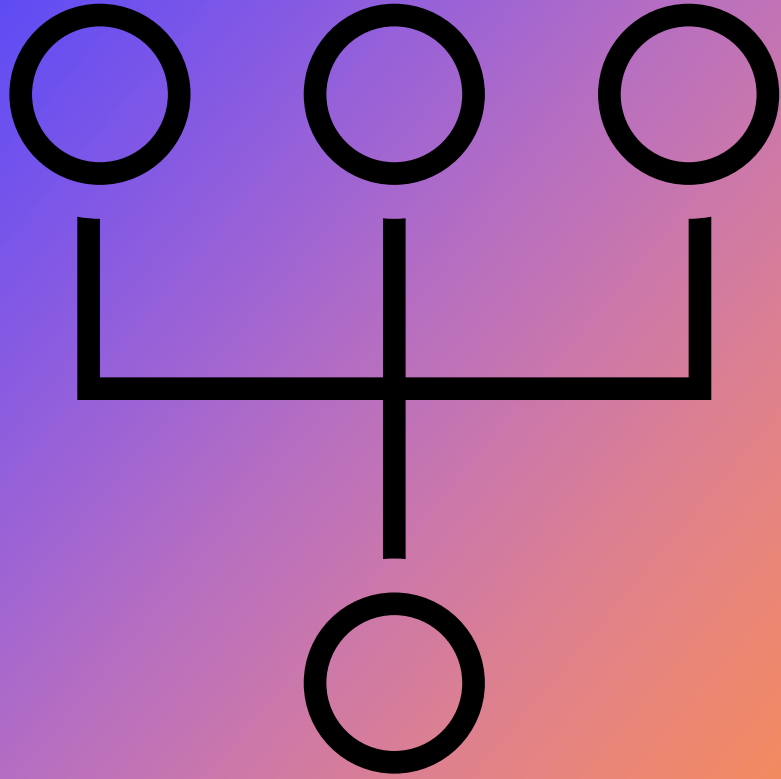
# Extra: Comandos de Linux

Algunos comandos útiles en Linux son

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a list of Linux commands with their functions explained in Spanish.

```
$ ls // Enlista el contenido de una carpeta
$ cd // Cambia a la carpeta que indiquemos
$ cd .. // Vuelve a la carpeta anterior
$ pwd // Muestra el path de la carpeta de trabajo actual
$ mkdir // Crea una nueva carpeta
$ rmdir // Elimina una carpeta o path
$ cp // Copia archivos y carpetas, incluyendo el contenido
$ mv // Mueve o renombra archivos y carpetas
$ touch // Crea un nuevo archivo vacío
$ nano // Edita un archivo con el editor de texto Nano
$ wget // Descarga archivos a partir de un URL
```





# BÁSICOS DE GIT

# Git

## Creando un repositorio

Generalmente, podemos obtener un repositorio de dos maneras:

1. Tomando un repositorio local que no esté bajo control de versiones, y lo convertimos en un repositorio de Git.
2. O, podemos **clonar** un repositorio existente de Git desde cualquier lado.

# Git

## Creando un repositorio

Desde el Git Bash:

```
// Primero, nos movemos a la carpeta donde vamos a trabajar
$ cd C:/Users/user/Desktop

// Luego, creamos la carpeta que almacenará el repositorio
$ mkdir "mi_proyecto"

// Nos movemos a esa carpeta
$ cd mi_proyecto

// Una vez ahí, escribimos:
$ git init
```


# Git

## Creando un repositorio

El comando `git init` crea una subcarpeta llamada `.git`, que contiene todos los archivos del repositorio necesarios. Hasta ese punto, aún no se registra nada en el proyecto.

Si queremos comenzar a registrar el control de versiones de los archivos, podemos añadir un nuevo archivo a la carpeta.

En terminal:




```
$ touch prueba.txt
```

# Git

## Creando un repositorio

Después, podemos editar el archivo en la misma terminal utilizando el editor Nano.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt character is a blue dollar sign, followed by the command 'nano prueba.txt' in blue text.

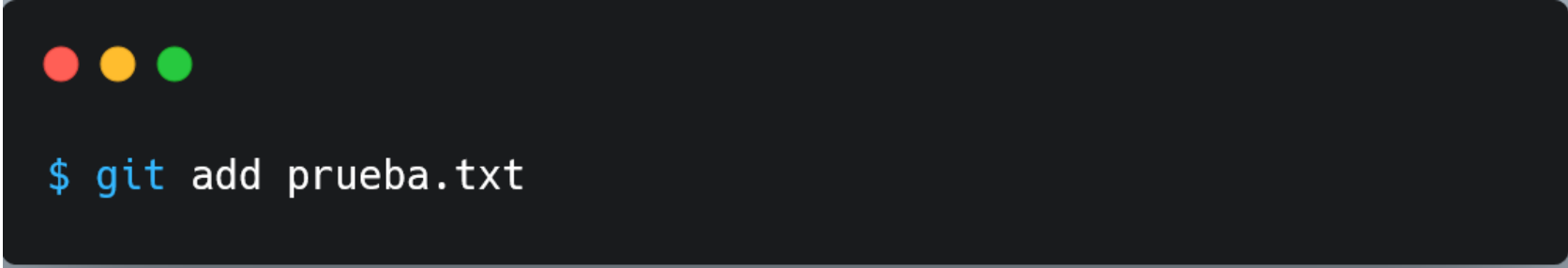
```
$ nano prueba.txt
```

Ojo, porque este editor admite únicamente entrada de teclado, no responde al mouse. Para salir oprimimos Ctrl + X y tecleamos Y para guardar los cambios. Nos pedirá confirmar con Enter y ya está.

# Git

## Creando un repositorio

Ahora podemos empezar a registrar los cambios al archivo. En terminal, lo hacemos con el comando `git add <nombre del archivo>`

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The command `$ git add prueba.txt` is entered in a light blue monospace font.

```
$ git add prueba.txt
```

Para verificar que se registró en el staging area podemos escribir `git status`

# Git

## Creando un repositorio



```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   prueba.txt
```

# Git

## Creando un repositorio

Recordemos que la siguiente etapa del flujo es asegurarnos que se almacenen los cambios, a través del **commit**, entonces lo siguiente es escribir `git commit` en la terminal.

Esto abrirá automáticamente el editor de texto que seleccionamos en la instalación, donde **necesitamos** escribir el mensaje del commit, con la finalidad de hacer una descripción de los cambios que se han realizado en esa sesión (referencia personal).

Una vez que hayamos escrito el mensaje, guardamos el archivo y automáticamente volvemos al Git Bash.



```
$ git commit
```



# Git

## Creando un repositorio

COMMIT\_EDITMSG ✕

```
1 Añado el primer archivo a mi repositorio.
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 #
5 # On branch master
6 #
7 # Initial commit
8 #
9 # Changes to be committed:
10 #   new file:   prueba.txt
11 #
12
```



```
$ git commit
```

```
[master (root-commit) b9a8f31] Añado el primer archivo a mi
repositorio.
```

```
1 file changed, 3 insertions(+)
create mode 100644 prueba.txt
```



# INTRODUCCIÓN A GITHUB

# GitHub

## Introducción a GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos en la nube utilizando el sistema de control de versiones Git.

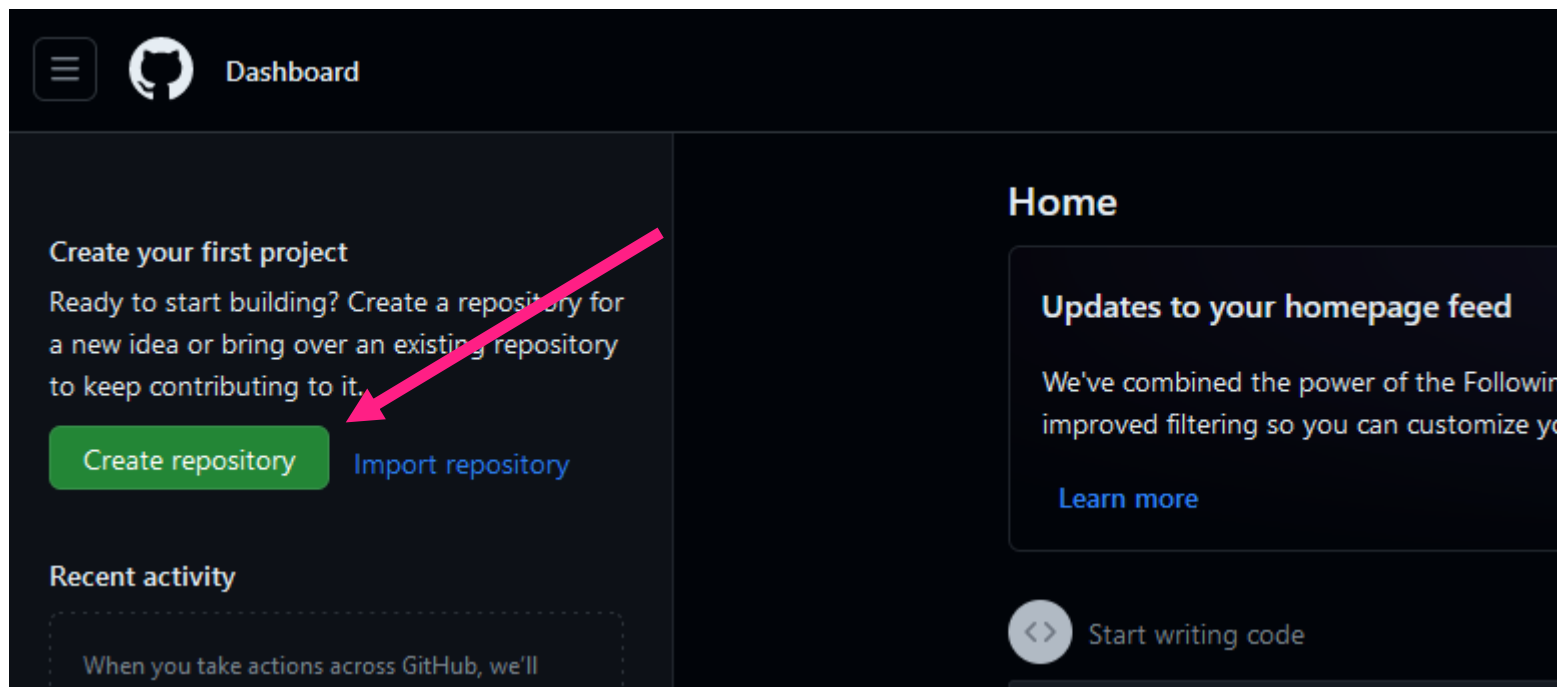
Es necesario crear una cuenta si eres un nuevo usuario:

[Crear cuenta en GitHub](#)

Una vez creada la cuenta podemos hacer un repositorio en la nube.

# GitHub

## Paso 1



# GitHub

## Paso 2

Hay que rellenar los campos de **Repository name**, **Description**, de preferencia hacer click en **Add a README file** y finalmente hacer click en **Create Repository**.

**Create a new repository**

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

Required fields are marked with an asterisk (\*).

Owner \* / Repository name \*

poputonga / repo-baby

repo-baby is available.

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

Description (optional)

Mi primer repo en github :D

☒ 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

License: None

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

This will set `main` as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

[Create repository](#)

# GitHub

## Paso 3

Por motivos de seguridad, es necesario configurar un token de acceso personal.

En GitHub:

Settings > Developer settings > Personal Access Tokens

Llenar los datos requeridos:

Nota (para qué usarás el token)

Expiración

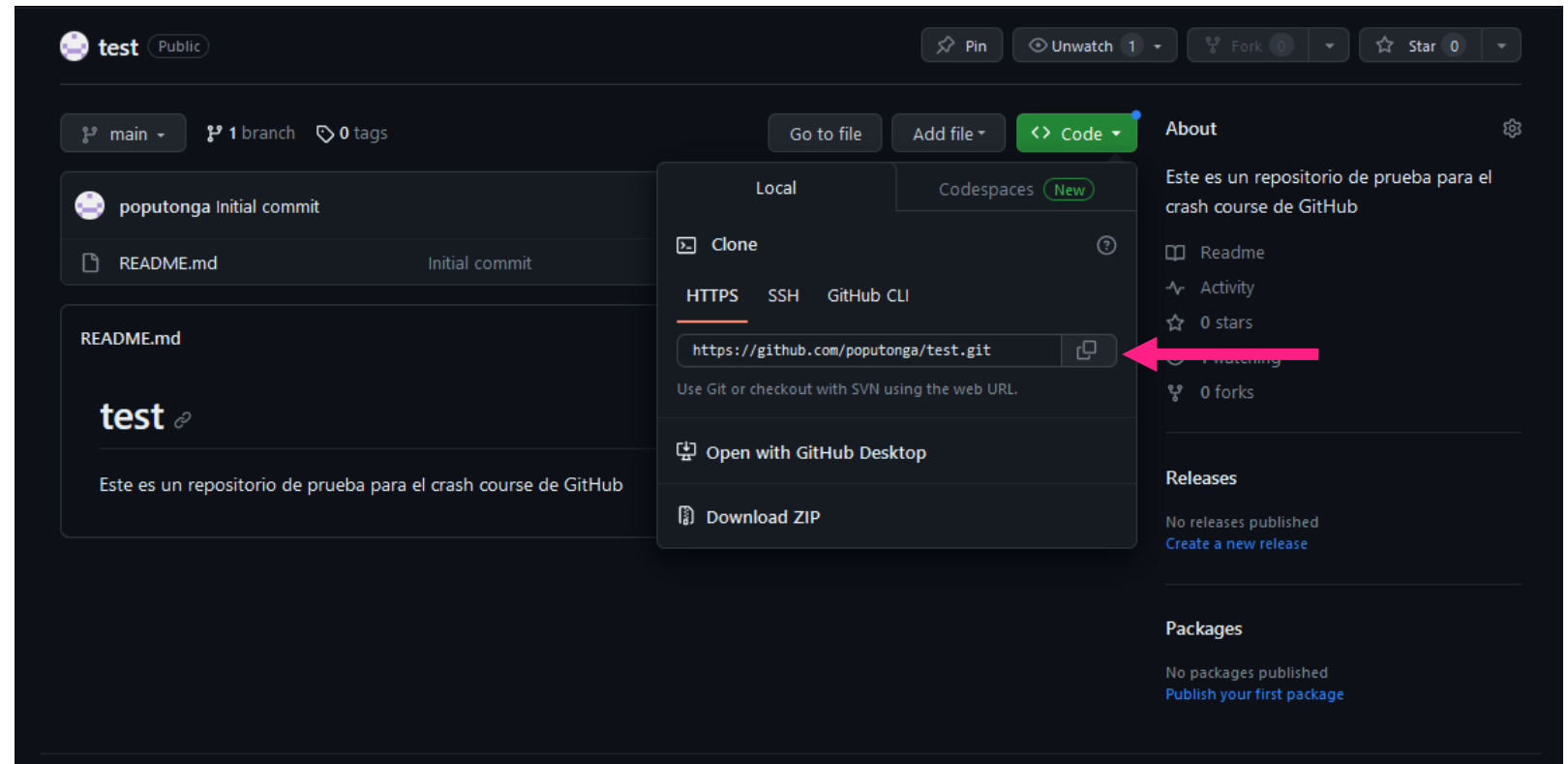
Scopes (seleccionar repo)

**IMPORTANTE:** Hay que guardar el token en un lugar seguro.  
No volverás a tener acceso después.

# GitHub

## Paso 4

Copiamos el  
URL del repositorio



# GitHub

## Paso 5

Es hora de volver a nuestra computadora. Como antes, creamos una carpeta donde almacenaremos el repositorio y nos situamos ahí en el Git Bash.

Una vez ahí, escribimos `git clone <url del repo>`



```
$ git clone https://github.com/FranzValerio/test.git
Cloning into 'test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```



# GitHub



## Paso 6

¡Listo! Tenemos una copia idéntica al repositorio que está en la nube pero ahora de manera local.

Nos cambiamos a la carpeta del repositorio de git, que en este caso se llama test.

Ahora generemos un archivo de prueba para añadirlo al repo.



```
$ cd test  
$ touch prueba.py  
$ nano prueba.py
```

# GitHub



## Paso 7

Ya que tenemos el archivo, lo... ¡añadimos al staging area!

```
$ git add prueba.py
```

Podemos verificar el estado:

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   prueba.py
```

# GitHub

## Paso 8

El siguiente paso es hacer el commit, con su respectivo mensaje:



```
$ git commit
```

```
[main c185826] Añado un archivo al repositorio de github  
1 file changed, 1 insertion(+)  
create mode 100644 prueba.py
```

# GitHub

## Paso 8

Cuando hayamos realizado todos los cambios, es hora de subirlos a la nube. Para ello, utilizamos el paso final del ciclo de trabajo de Git: push.

Escribimos entonces:

```
$ git push origin main
```

```
Enumerating objects: 4, done.
```

```
Counting objects: 100% (4/4), done.
```

```
Delta compression using up to 12 threads
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 366 bytes | 366.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/FranzValerio/test.git
```

```
81f0abf..c185826  main -> main
```

# GitHub

## Paso 9

Si fue exitoso, podemos darle refresh a la página de nuestro repositorio en GitHub, donde vamos a encontrar que los cambios que se hicieron de manera local ¡ya se reflejan en la nube! 😊



A vertical bar on the left side of the slide with a gradient from orange at the top to blue at the bottom.

# GitHub

## Paso 10

Sea feliz.





# GitHub

## Contenido extra

Este tipo de entorno de trabajo virtual permite que no sólo una persona obtenga una carpeta de archivos con el mismo contenido que tú, sino que también puede contribuir a tu código.

De aquí surgen los conceptos de **tres & branches** y de **merge, fork y pull request**.

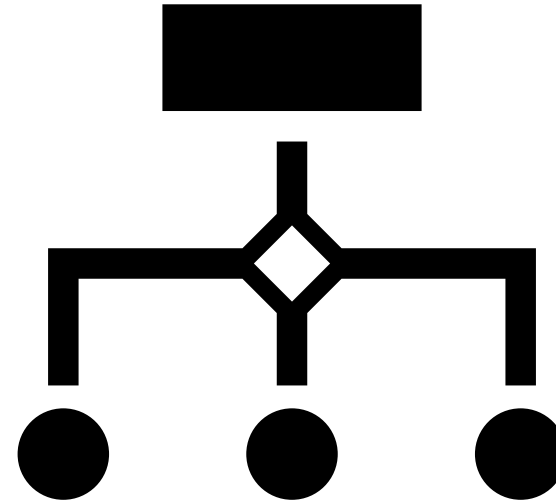
# GitHub

## Trees & Branches

Los **trees** (árboles) son la representación del sistema de archivos en un punto específico de la historia de un repositorio. Son una instantánea de todos los archivos y los directorios.

Por otro lado, las **branches** (ramas) son secuencias lineales de commits, que permiten desarrollos paralelos y separados.

Por default, main o master es la rama principal



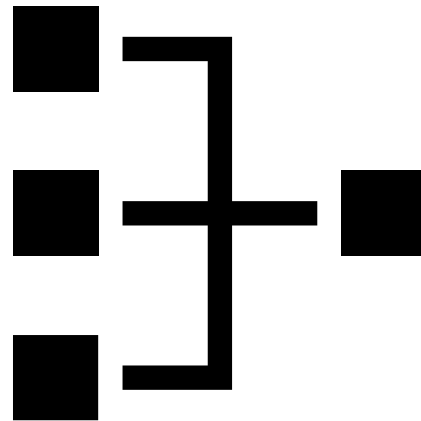


# GitHub

## Merge & Fork

Entonces, hacer un **merge** es el proceso de combinar cambios de una rama a otra. Por lo que resulta en un nuevo commit en la rama destino.

Mientras que un **fork** es la creación de una copia del repositorio original a tu cuenta personal de GitHub. Esto es muy útil porque permite modificar un proyecto sin afectar el original.

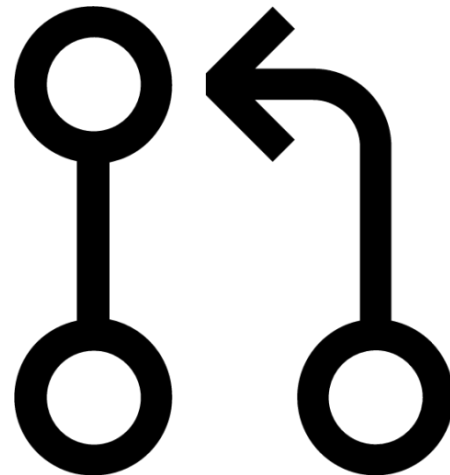


# GitHub

## Pull request

Finalmente, si un usuario trabaja en un repositorio ajeno, pero tiene una buena propuesta para modificar y mejorarlo, puede enviar una propuesta para incluir cambios de una rama de ese repositorio.

Esto facilita la revisión del código y la colaboración. Y se pueden discutir y revisar cambios antes de fusionar.





# Conclusiones

- El uso de VCS ha revolucionado la manera en que trabajamos y gestionamos proyectos de código.
- Git brinda flexibilidad y robustez inigualables, nos permite trabajar de manera distribuida.
- GitHub, por su parte, no solo sirve como una plataforma de alojamiento para proyectos gestionados con Git, sino que también fomenta la colaboración y el trabajo en equipo.

+



o



.



# GRACIAS