

Creación y uso de repositorios GITHUB con ayuda de GIT

JUAN DAVID RUGE GARZÓN

**Trabajo informe creado con la finalidad de explicar el uso de los comandos GIT reflejados
en uso GITHUB**

DOCENTE

WILLIAM ALEXANDER MATALLANA PORRAS

PROGRAMACIÓN II

Universidad de Cundinamarca Extensión Chía

INGENIERIA DE SISTEMAS Y COMPUTACIÓN

2025

Contenido

1. Introducción	3
1.1 Marco teórico	3
1.2. Desarrollo.....	3
2. Conclusiones	15
Referencias	16

1. Introducción

Git es una herramienta clave para la gestión de proyectos de código, especialmente cuando trabajamos en equipo. Este trabajo busca de cierta forma explicar de una manera sencilla y con ayuda de material fotográfico o videos, como usar comandos básicos de GIT para administrar repositorios en GITHUB. Aprenderemos a crear un repositorio desde 0, subir cambios, crear ramas y fusionar cambios, entre otras tareas comunes, con el fin de mantener un flujo de trabajo ordenado y eficiente. Ideal para aquellos que quieran tener una familiarización con GIT y así mismo mejorar manejo de proyectos en GITHUB

1.1 Marco teórico

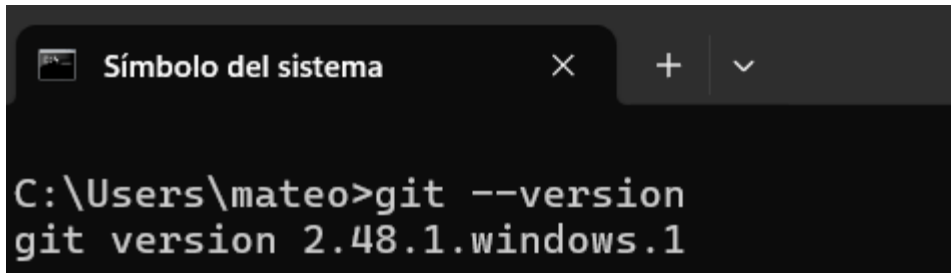
GIT es un sistema de control de versiones ampliamente utilizado en el desarrollo de software. El cual tiene como función principal rastrear cambios en archivos y coordinar el trabajo entre varias personas. Permite flexibilidad y autonomía.

GitHub es una plataforma en línea que utiliza GIT para alojar repositorios. Facilita la colaboración, dado que permite a los usuarios compartir, revisar y gestionar código.

IntelliJ es un entorno de desarrollo integrado (IDE) que se utiliza para creación de programas, gestión de proyectos, depuración y codificación.

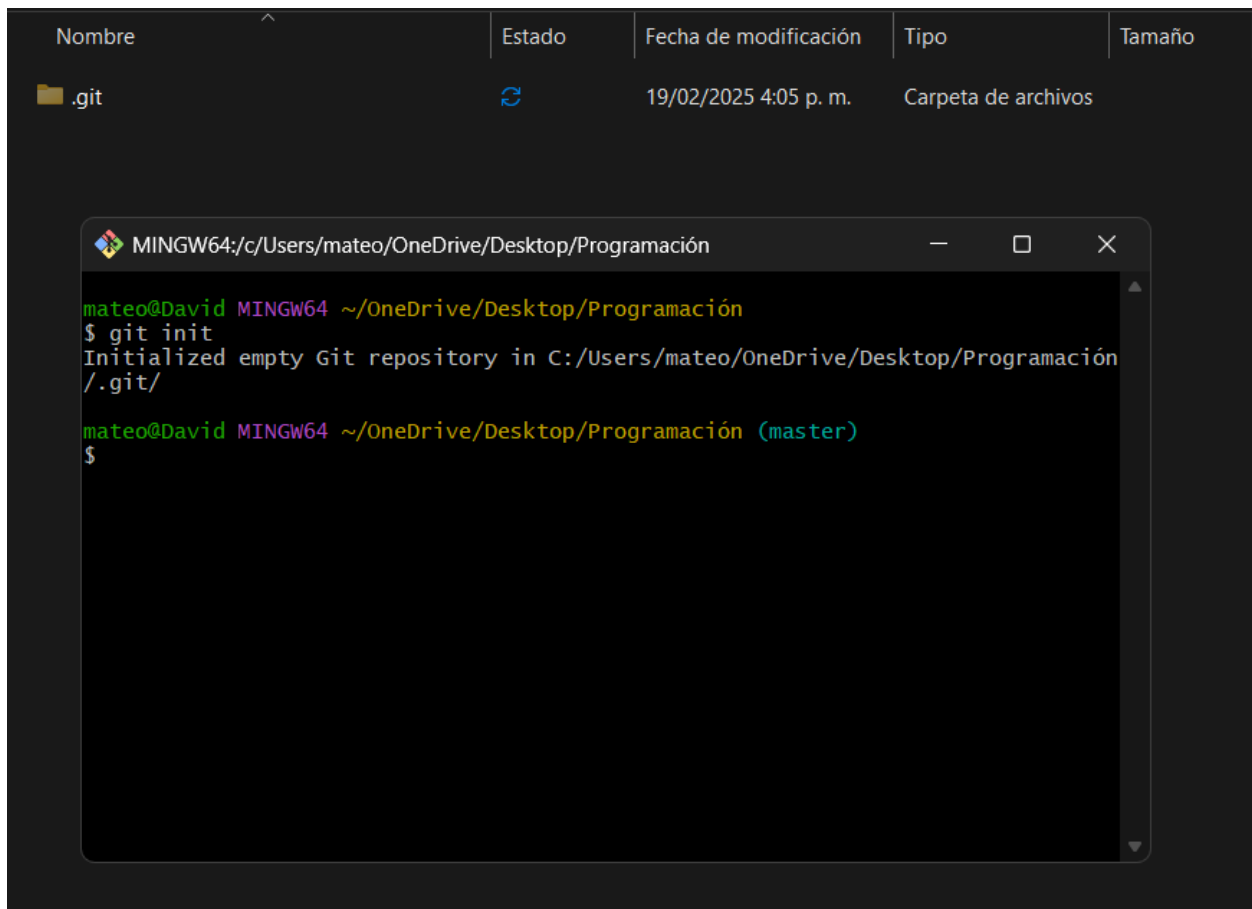
1.2. Desarrollo

Para comenzar con el uso de comandos básicos de GIT, tal deberá estar instalado en nuestro sistema y seguido a eso deberemos verificar que tal este configurado de una manera correcta con el comando **git --version** en una terminal o consola.



```
C:\Users\mateo>git --version
git version 2.48.1.windows.1
```

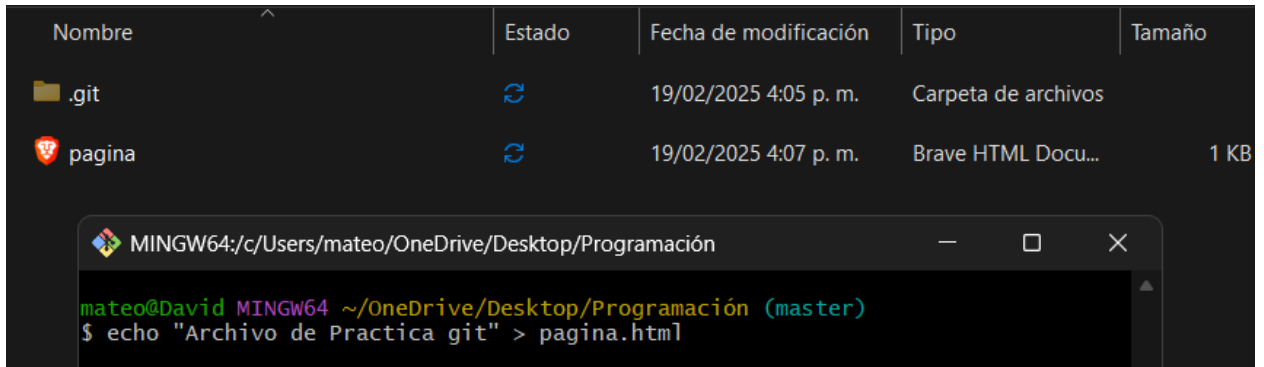
Este comando nos mostrará la versión de GIT instalada, asegurando así que esta listo para empezar. Ahora se iniciará a crear un repositorio local con el comando **git init** el cual inicializa un nuevo repositorio en la carpeta actual.



Nombre	Estado	Fecha de modificación	Tipo	Tamaño
.git		19/02/2025 4:05 p. m.	Carpeta de archivos	

```
MINGW64:/c/Users/mateo/OneDrive/Desktop/Programación
mateo@David MINGW64 ~/OneDrive/Desktop/Programación
$ git init
Initialized empty Git repository in C:/Users/mateo/OneDrive/Desktop/Programación/.git/
mateo@David MINGW64 ~/OneDrive/Desktop/Programación (master)
$
```

Este comando creará una carpeta oculta llamada **.git** donde GIT almacenará toda información relacionada con historial y cambios del proyecto, ahora como practica se creará un archivos de ejemplo llamado *pagina.html*.



Con esto ya se tiene un archivo listo para ser gestionado con git.

Configuración GIT

Para empezar a trabajar con git primero deberemos configurarlo de tal forma que quede anclado a nuestro perfil y email de github, para esto deberemos usar los siguientes comandos:

Git config --list cuya función es enlistar todas las configuraciones realizadas en git

```
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=schannel
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=DavidAiko
user.email=kenx@fall.cat
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

Git config --global user.email & git config --global user.name con estos dos comandos realizaremos una conexión de git a nuestro perfil y mail de github, donde deberemos colocarlos seguidos de un espacio.

```
mateo@David MINGW64 ~/OneDrive/Desktop/Programación (master)
$ git config --global user.name DavidAiko

mateo@David MINGW64 ~/OneDrive/Desktop/Programación (master)
$ git config --global user.email kenx@fall.cat
```

Lo cual al enlistar saldrá dentro de las configuraciones

```
user.name=DavidAiko  
user.email=kenx@fall.cat
```

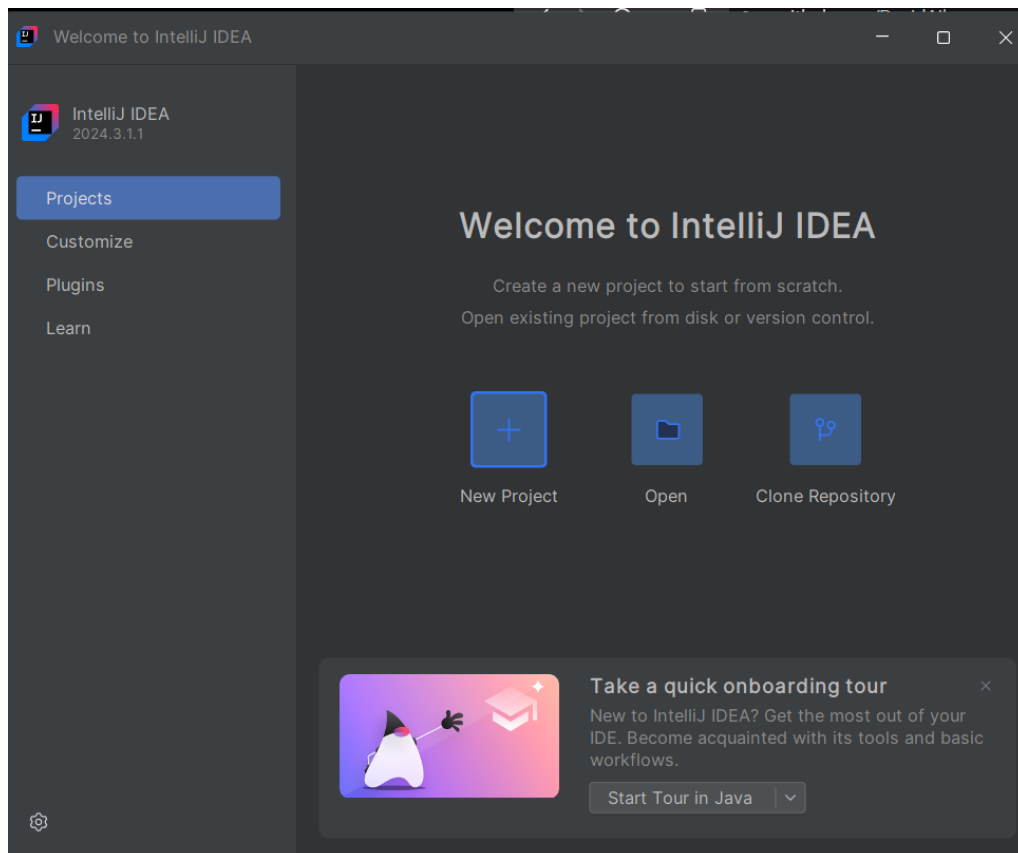
En caso de que queramos remover tal configuración deberemos usar los siguientes comandos

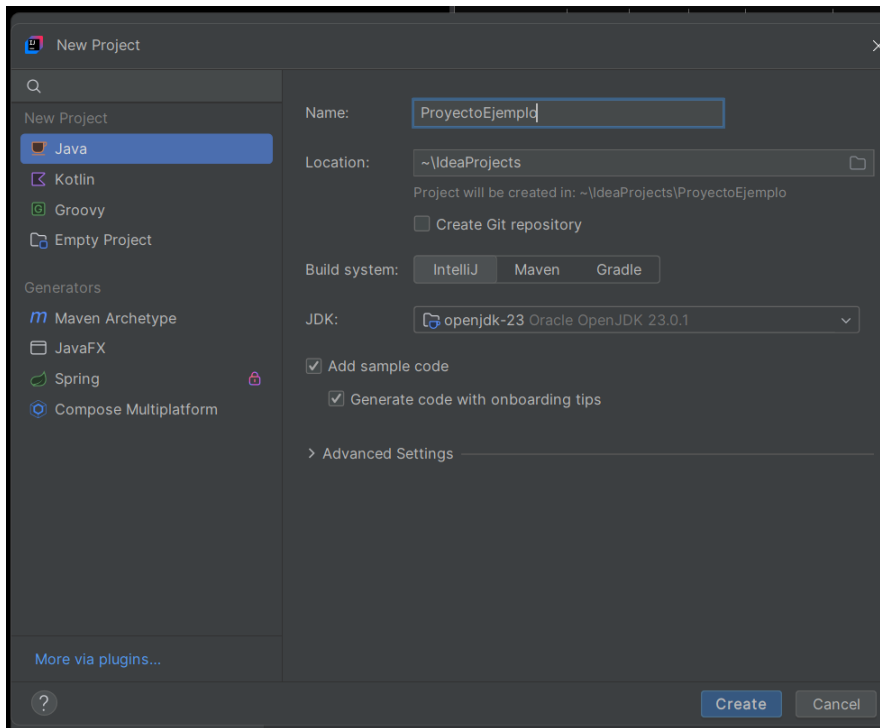
Git config --global --unset user.name

Git config --global --unset user.email

Proyectos en IntelliJ

Vamos a crear un proyecto JAVA en IntelliJ





Y sincrónicamente 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.](#)

Required fields are marked with an asterisk ().*

Owner * DavidAiko / **Repository name *** RepositorioEjemplo
✔ RepositorioEjemplo is available.

Great repository names are short and memorable. Need inspiration? How about **congenial-octo-sniffle** ?

Description (optional)

☒ **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**

Copiamos el comando que GitHub nos brinda y lo pegamos en la terminal del proyecto en IntelliJ, lo cual hará que nuestro proyecto en IntelliJ cree una conexión con el repositorio en GitHub

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

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

Lo cual al colocarlo en la terminal de IntelliJ se ve de la siguiente manera:

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> echo "# RepositorioEjemplo" >> README.md
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git init
Initialized empty Git repository in C:/Users/mateo/IdeaProjects/ProyectoEjemplo/.git/
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git add README.md
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git commit -m "first commit"
[master (root-commit) 96102d3] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch -M main
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git remote add origin https://github.com/DavidAiko/RepositorioEjemplo.git
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 247 bytes | 123.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/DavidAiko/RepositorioEjemplo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo>
```

Con esto ya tenemos el proyecto local anclado con el repositorio de GitHub, ahora solo queda crear un código nosotros mismos y subirlo a continuación.


```

1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          System.out.println("Calculadora para el Proyecto Ejemplo");
8
9          System.out.print("Ingrese el primer número: ");
10         double num1 = scanner.nextDouble();
11         System.out.print("Ingrese el segundo número: ");
12         double num2 = scanner.nextDouble();
13
14         // Suma
15         double suma = num1 + num2;
16         System.out.println("La suma es: " + suma);
17
18         // Resta
19         double resta = num1 - num2;
20         System.out.println("La resta es: " + resta);
21
22         scanner.close();
23     }
24 }

```

Con el comando **git add .** añadiremos todos los archivos que tenemos de forma local al repositorio anteriormente conectado, pero no se subirán. Para subirlos deberemos generar un commit el cual llegará como pseudo-actualización del repositorio acompañado de los archivos con el comando **git commit -m "mensaje"**

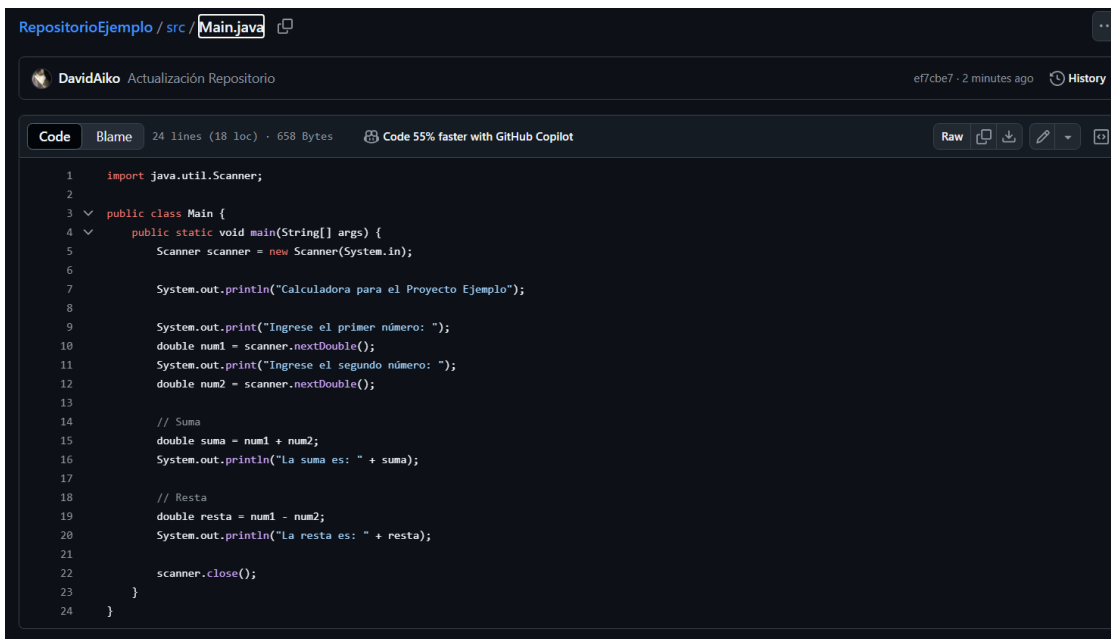
```

PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git commit -m "Actualización Repositorio"
[main ef7cbe7] Actualización Repositorio
7 files changed, 87 insertions(+)
create mode 100644 .gitignore
create mode 100644 .idea/.gitignore
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 ProyectoEjemplo.iml
create mode 100644 src/Main.java
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo>

```

Y finalmente subiremos los archivos con el comando **git push origin main** lo cual “EMPUJARÁ” todos los archivos del local al repositorio acompañado del commit

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git push origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 1.87 KiB | 383.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/DavidAiko/RepositorioEjemplo.git
96102d3..ef7cbe7  main -> main
```



The screenshot shows the GitHub web interface for the repository 'RepositorioEjemplo' by user 'DavidAiko'. The file 'Main.java' is selected, showing its contents. The file is 24 lines long, 18 lines of code, and 658 bytes. It contains a Java program that uses the Scanner class to read two numbers and calculate their sum and difference. The commit hash 'ef7cbe7' is shown, along with the time '2 minutes ago' and a 'History' link. The code is as follows:

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          System.out.println("Calculadora para el Proyecto Ejemplo");
8
9          System.out.print("Ingrese el primer número: ");
10         double num1 = scanner.nextDouble();
11         System.out.print("Ingrese el segundo número: ");
12         double num2 = scanner.nextDouble();
13
14         // Suma
15         double suma = num1 + num2;
16         System.out.println("La suma es: " + suma);
17
18         // Resta
19         double resta = num1 - num2;
20         System.out.println("La resta es: " + resta);
21
22         scanner.close();
23     }
24 }
```



The screenshot shows the file list for the repository 'RepositorioEjemplo' by user 'DavidAiko'. The table lists the files and folders in the repository, along with the commit hash 'ef7cbe7' and the time '2 minutes ago'. There are 2 commits. The files are:

File	Commit	Time
.idea	Actualización Repositorio	2 minutes ago
src	Actualización Repositorio	2 minutes ago
.gitignore	Actualización Repositorio	2 minutes ago
ProyectoEjemplo.iml	Actualización Repositorio	2 minutes ago
README.md	first commit	12 minutes ago

Below the table, there is a section for the 'README' file, which is currently empty.

Creación de Ramas

Una rama en GitHub es una versión independiente del repositorio principal el cual permite trabajar sin afectar el código principal, en este caso la rama **main**, se usa generalmente para aislar cambios, corregir errores o tener un flujo más estable.

A continuación, identificaremos como primer paso las ramas existentes en el repositorio con el comando **git branch**

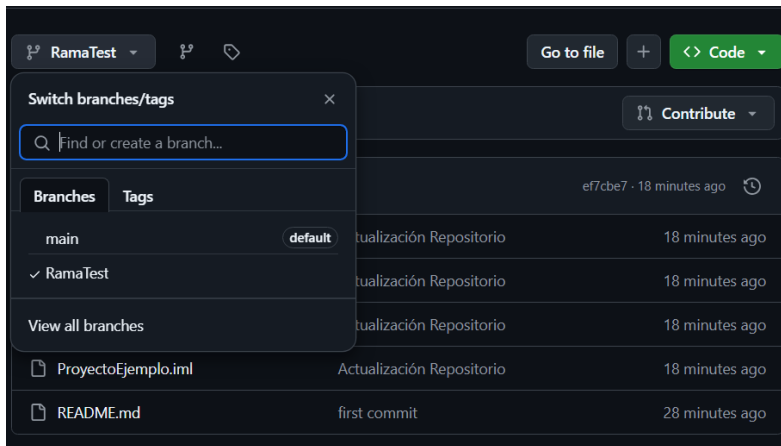
```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch
* main
```

En este caso detecto únicamente la rama main la cual es la rama principal pero para que podamos tener más ramas usaremos el comando **git switch -c** comando el cual tiene el sufijo -c que hace la función **CREAR**

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git switch -c RamaTest
Switched to a new branch 'RamaTest'
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch
* RamaTest
  main
```

Al crear nuestra rama **RamaTest** y ejecutar el comando **git branch** ahora podemos identificar que se encuentran dos ramas creadas y estamos ahora mismo en la rama **RamaTest**, en caso de que queramos cambiar de rama podemos usar el comando **git switch NOMBRE** para ubicarnos encima de una rama diferente

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch
  RamaTest
* main
```



Pull Request

Es un cierto tipo de solicitud para fusionar los cambios de una Rama 1 con una Rama 2.

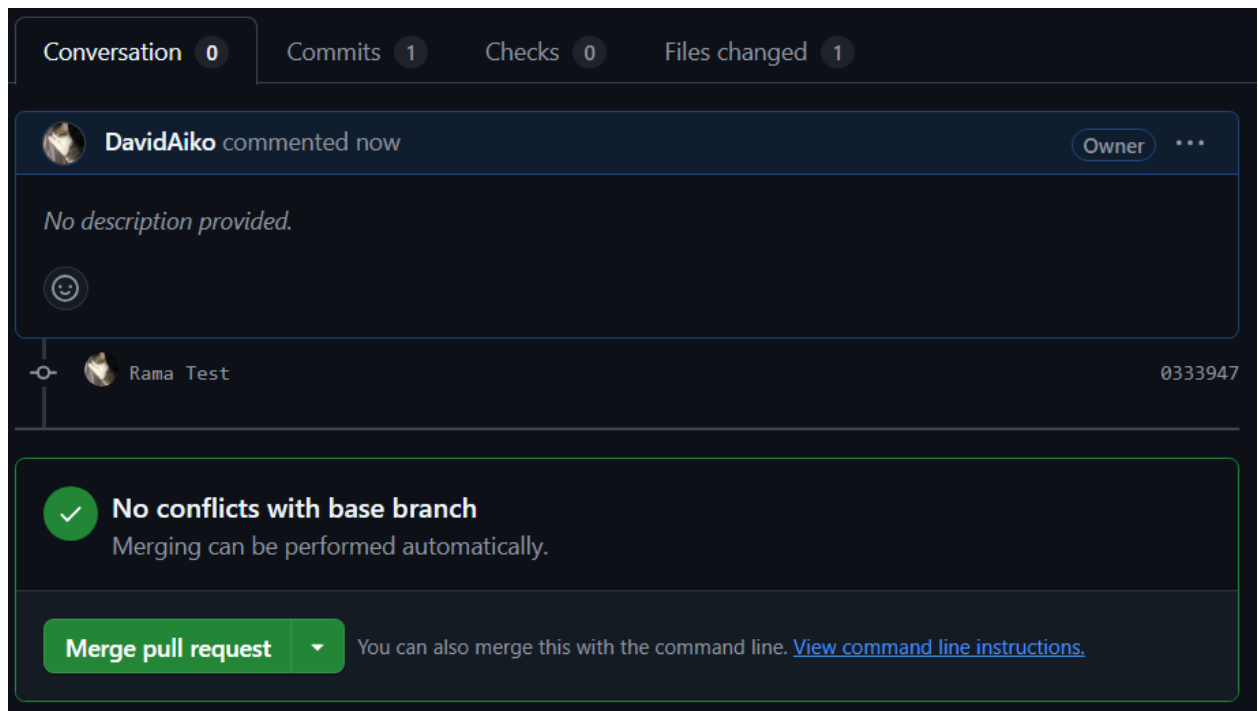
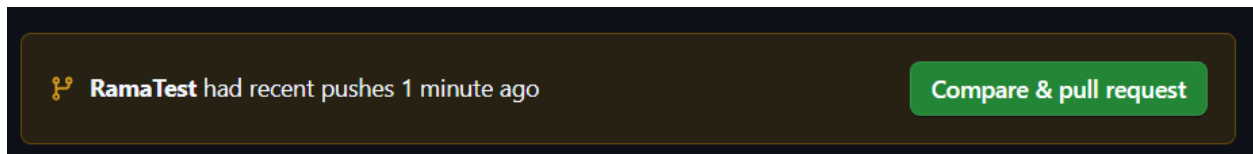
Es decir, proponer cambios, revisarlos e integrarlos al código base. A continuación, generaremos un PR desde la rama **RamaTest** la cual modifiko el Main de tal manera

```
1  
2  
3 public class Main {  
4     public static void main(String[] args) {  
5         System.out.println("Test PullRequest");  
6     }  
7 }
```

Y haremos push del código actual hacia la RamaTest

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git add .  
warning: in the working copy of 'src/Main.java', LF will be replaced by CRLF the ne  
xt time Git touches it  
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git commit -m "Rama Test"  
[RamaTest 0333947] Rama Test  
1 file changed, 2 insertions(+), 19 deletions(-)  
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git push origin RamaTest  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (4/4), 349 bytes | 349.00 KiB/s, done.  
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)  
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.  
To https://github.com/DavidAiko/RepositorioEjemplo.git  
ef7cbe7..0333947 RamaTest -> RamaTest
```

En nuestro GitHub saldrá un nuevo botón el cual tiene la función de comparar el código y generar una PullRequest



El botón merge realizará una fusión entre ambas ramas y convertirá los archivos de la rama **RamaTest** en archivos de la rama **MAIN**. En caso de que queramos borrar la rama a causa de que ya no la necesitamos o no se le tiene uso, utilizamos el comando **git branch -d NOMBRE** el cual tiene el prefijo -D para borrar la rama.

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch -d RamaTest
Deleted branch RamaTest (was 0333947).
```

Trabajo en Ramas Remotas

Uso del **git fetch --all**

Este comando se utiliza para poder obtener todos los cambios de los repositorios remotos SIN NECESIDAD de fusionarlos con la rama actual. Utilizando un método en el que descarga cambios de commits, ramas y tags de los remotos como origin.

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git fetch --all
```

Uso del **git branch -r**

Este comando se utiliza para mostrar todas las ramas remotas disponibles en el repo local.

```
PS C:\Users\mateo\IdeaProjects\ProyectoEjemplo> git branch -r
origin/HEAD -> origin/main
origin/RamaTest
origin/main
```

Uso de **git merge** y **git rebase**

Ambos comandos se utilizan para integrar los cambios de una rama en otra, aunque de manera diferente

Git merge por su parte combina cambios de una rama en la rama actual, creando un commit de fusión que une ambas.

Git rebase por otro lado replica los commits de una rama actual sobre otra rama, lo que implica una reescritura de historial.

Repositorio usado para el ejemplo:

<https://github.com/DavidAiko/RepositorioEjemplo/tree/main>

2. Conclusiones

IntelliJ IDEA hace que el trabajar con Git y GitHub sea más sencillo y eficiente. Gracias a que con sus herramientas integradas se puede crear y clonar repositorios, manejar ramas, hacer commits o sincronizar cambios con unos simple comandos. Esto permite al programador enfocarse en escribir código sin preocuparse por desorganización estructural de sus repositorios, logrando así mismo que la colaboración en proyectos logre ser más fluida y organizada.

Referencias

1. Apuntes tomados en clase. (14 de febrero de 2025). Notas personales sobre el uso de Git y GitHub en IntelliJ IDEA.
2. Aprendizaje Autodidacta en YouTube. (15 y 16 de febrero 2025). Canales educativos de programación, como:
 - <https://www.youtube.com/watch?v=vlCXdvvgiE0>
 - <https://www.youtube.com/watch?v=44ziZ12rJwU>
 - <https://www.youtube.com/watch?v=3XlZWpLwvvo>
3. GitHub Guides. (2025). Recuperado de <https://guides.github.com/>
4. Uso de Inteligencia Artificial (IA). (2025). Para elaboración de algunos tecnicismos y marco teórico, se utilizo asistencia de IA (DeepSeek) para aclarar conceptos y mejorar redacción.