

Actividad 1 Programación

Mónica Yulihet Poveda Carrasco

Profesor

William Alexander Matallana Porras

Universidad de Cundinamarca, Extensión Chía

Faculta de Ingenieria

Programación I

2025

Tabla de Contenido

| | |
|----------------------------------|----|
| Introducción | 3 |
| Objetivo | 3 |
| Desarrollo | 4 |
| Conclusión | 37 |
| Referencias bibliográficas | 37 |

Introducción

El presente trabajo busca explicar el uso de los comandos de *Git*, a través de un ejercicio práctico desarrollado en *IntelliJ IDEA* (entorno de desarrollo integrado para el desarrollo de programas informáticos) utilizando Java 21. En el cual evidencia el funcionamiento y su aplicación dentro de la interfaz del código de comandos como “*git status*”, “*git Branch*”, “*git commit -m “ ”*”, entre otros. Adicionalmente se muestra el proceso de almacenamiento del código en un repositorio de GitHub.

Git Hub, una plataforma web de desarrollo colaborativo basada en Git, un sistema de control de versiones (SCV), que permite realizar un seguimiento de los cambios en los archivos. Esta herramienta facilita el trabajo en equipo al permitir almacenar, compartir y colaborar en proyectos de código de manera eficiente.

Objetivo

Ejemplificar el uso de los comandos de *git* a través de un ejercicio práctico, utilizando un código desarrollado en IntelliJ IDEA con java 21, y realizar su almacenamiento en un repositorio de GitHub.

Desarrollo

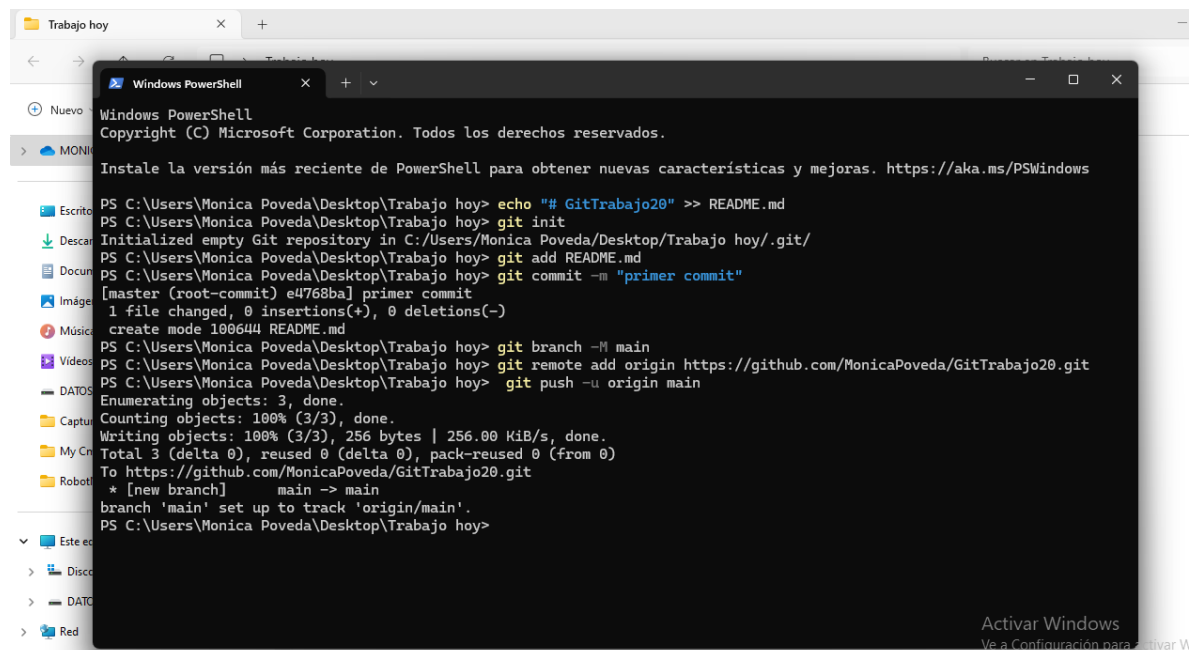
Para empezar a desarrollar nuestro código en IntelliJ IDEA se debe utilizar un programa que permita integrar directamente el proyecto en un repositorio, como lo es GitHub.

GitHub es una plataforma de desarrollo de software basada en la web que permite a los desarrolladores almacenar, compartir y colaborar en proyectos de software. Es la herramienta más popular para trabajar con Git, un sistema de control de versiones (SCV) que permite llevar un registro de los cambios realizados en el código fuente a lo largo del tiempo.

Inicialmente creamos una carpeta para asociarla a nuestro repositorio.

Una vez creada la carpeta, nos dirigimos al repositorio creado, allí se desplegará una línea de comandos.

Una vez copcada esta línea, nos dirigimos a la carpeta que habíamos creado anteriormente y abrimos la terminal. En la terminal pegamos el link que nos arrojó el repositorio.



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> echo "# GitTrabajo20" >> README.md
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git init
Initialized empty Git repository in C:/Users/Monica Poveda/Desktop/Trabajo hoy/.git/
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git add README.md
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git commit -m "primer commit"
[master (root-commit) e4768ba] primer commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git branch -M main
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git remote add origin https://github.com/MonicaPoveda/GitTrabajo20.git
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 256 bytes | 256.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MonicaPoveda/GitTrabajo20.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\Monica Poveda\Desktop\Trabajo hoy>
  
```

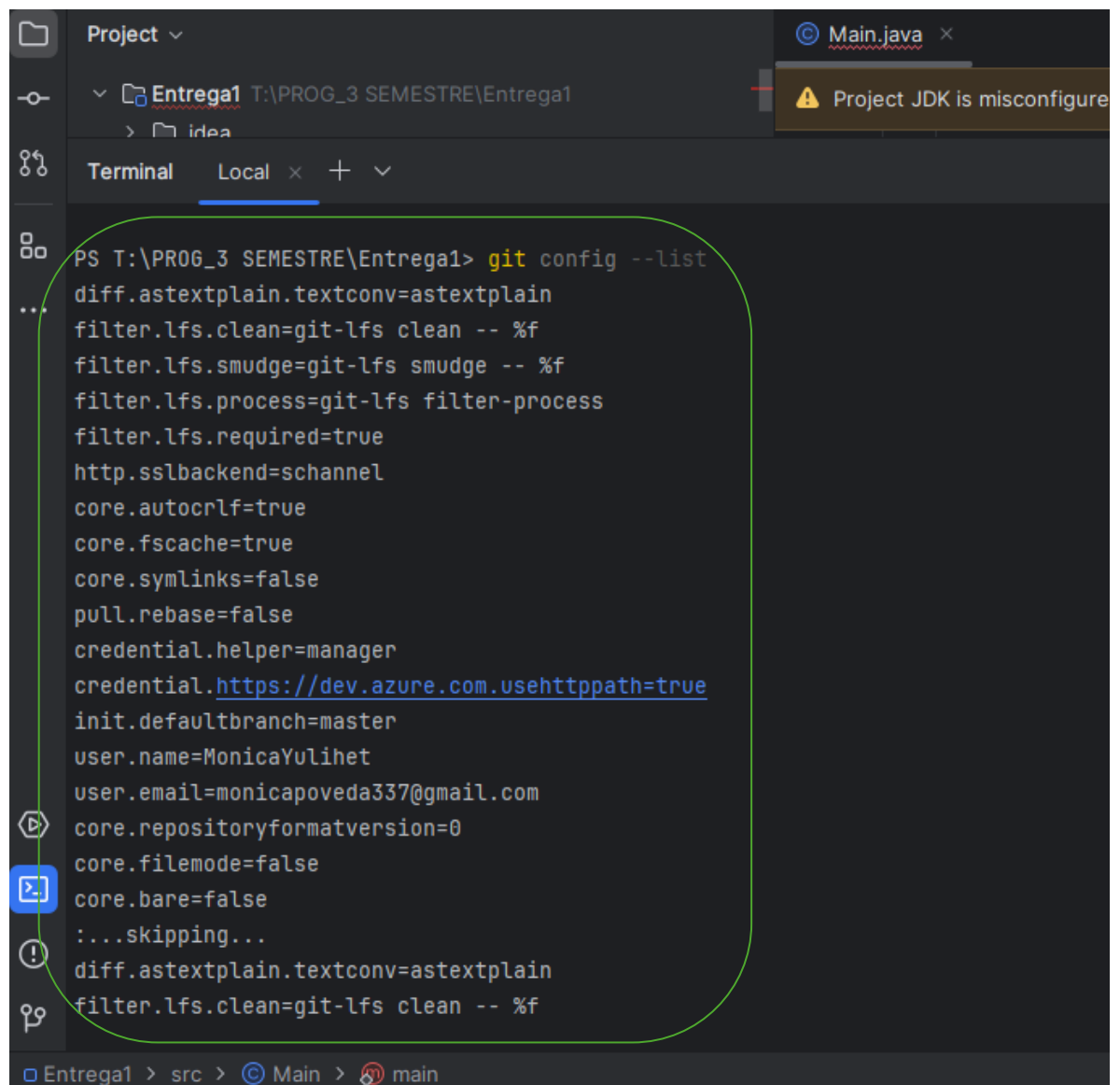
Nota. La imagen muestra como es asociado el repositorio en la carpeta, subiendo así la carpeta al repositorio con la extensión .README.

Después de esto abrimos un proyecto nuevo en IntelliJ IDEA.

Al abrirlo, nos dirigiremos a la terminal a configurar el usuario y el correo con la cual se vinculará el proyecto.

Comando *git config --global --list*

Este comando te muestra la configuración global de Git. Muestra una lista de todos los valores configurados globalmente, como el nombre de usuario, correo electrónico, editor, entre otros. Usar *--global* significa que se está viendo o modificando la configuración para todos los repositorios del local.



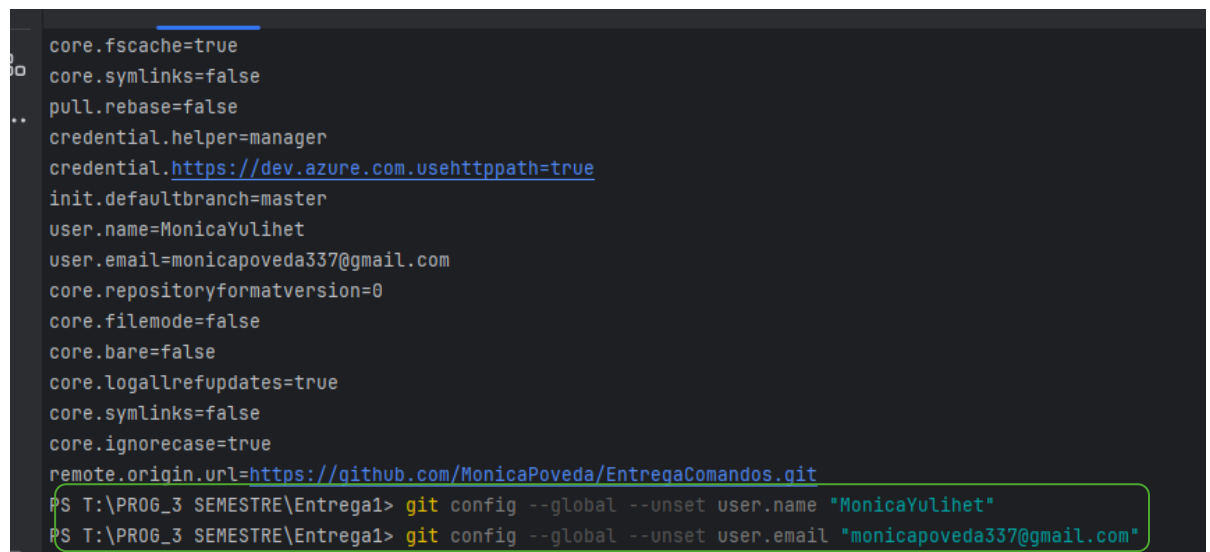
```
PS T:\PROG_3 SEMESTRE\Entrega1> git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=sschannel
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=MonicaYulihet
user.email=monicapoveda337@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
...skipping...
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
```

Comando *git config --global --unset user.name (..)*

Este comando elimina la configuración global del nombre de usuario en Git. Si previamente se configuro algún *user.name*, lo elimina de la configuración global.

Comando *git config --global --unset user.email (..)*

Similar al anterior, este comando elimina la configuración global del correo electrónico en Git, eliminando el *user.email* que se requiera eliminar. Esto también se aplica solo a la configuración global, no a la local.



```

core.fsckcache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=MonicaYulihet
user.email=monicapoveda337@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/MonicaPoveda/EntregaComandos.git
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global --unset user.name "MonicaYulihet"
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global --unset user.email "monicapoveda337@gmail.com"

```

Comando *git config --global user.name (..)*

Este comando configura globalmente el nuevo nombre de usuario para los commits de Git.

Comando *git config --global user.email (..)*

Este comando establece globalmente el nuevo correo electrónico que Git que se usará para asociarlo con los commits.

```

core.fsckcache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=MonicaYulihet
user.email=monicapoveda337@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/MonicaPoveda/EntregaComandos.git
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global --unset user.name "MonicaYulihet"
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global --unset user.email "monicapoveda337@gmail.com"
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global user.name "MonicaPoveda"
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global user.email "monicapoveda337@gmail.com"
PS T:\PROG_3 SEMESTRE\Entrega1> git config --global --list
user.name=MonicaPoveda
user.email=monicapoveda337@gmail.com
PS T:\PROG_3 SEMESTRE\Entrega1>

```

Después de haber realizado la configuración de usuario y correo, se procede a asegurar que la configuración se haya realizado correctamente, esto a través del comando anterior mencionado *git config --global --list* quien mostrará el usuario y email que se establecio. Para después iniciar con nuestro proyecto utilizando el comando *git init*.

Comando *git init*

El comando *git init* permite crear un nuevo repositorio de Git. El cual permite convertir un proyecto existente sin extensión a un repositorio de Git, o generalmente para iniciar un nuevo repositorio vacío. Generalmente este comando suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar el comando *git init*, Git crea un directorio oculto llamado *.git* el cual almacena todos los elementos que se trabajen dentro del proyecto. Este directorio oculto *.git* es lo que separa un directorio normal de un repositorio de Git. Como se ve a continuación.

git init en la terminal

A screenshot of a Windows command prompt terminal window. The prompt is 'PS T:\PROG_3 SEMESTRE\Entrega1>'. The first command entered is 'git config --global --list', which outputs 'user.name=MonicaPoveda' and 'user.email=monicapoveda337@gmail.com'. The second command entered is 'git init', which is highlighted with a green box. A green arrow points from this box to the email address in the previous output. The output for 'git init' is 'Reinitialized existing Git repository in T:/PROG_3 SEMESTRE/Entrega1/.git/'. The prompt returns to 'PS T:\PROG_3 SEMESTRE\Entrega1>'.

Nota. La imagen incorporada evidencia el *git init* en la terminal del proyecto de IntelliJ IDEA . Fuente: Elaboración propia (2025).

El desarrollo de un proyecto generalmente gira en torno a un patrón de instrucción: edición, preparación y confirmación. Primero, se editan los archivos en el directorio de trabajo, en este caso, en IntelliJ IDEA. Por lo tanto, se es necesario saber el estado actual del directorio de trabajo y del área de preparación de Git, esto a través del comando *git status*; El cual es como un resumen que indica qué cambios se han realizado para ser guardados nuevamente, y qué archivos nuevos aún no están siendo rastreados por Git. Sin embargo, vale la pena señalar que *git status* no brinda información sobre el historial de los cambios previos, es decir, no informa qué *commits* se han hecho ni qué se ha confirmado anteriormente. (ver imagen acontinuacion)

Figura 4.

Visualización de archivos no rastreados con el comando git status.


```

PS T:\PROG_3 SEMESTRE\Entrega21> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .idea/
        Entrega21.iml
        src/

nothing added to commit but untracked files present (use "git add" to track)

```

Nota. La imagen muestra la salida del comando *git status* en la terminal, indicando que hay archivos y carpetas no rastreados por Git. En este caso, se destacan tres elementos: *.gitignore*, que es un archivo utilizado para excluir ciertos archivos del control de versiones. *.idea/*, que es una carpeta creada por IntelliJ IDEA que contiene configuraciones del proyecto. Y *src/*, que es la carpeta que contiene el código fuente del proyecto. Por lo tanto, sugiere utilizar el comando *git add* . para agregar estos archivos al área de preparación y así incluirlos en el próximo *commit* . Fuente: Elaboración propia (2025).

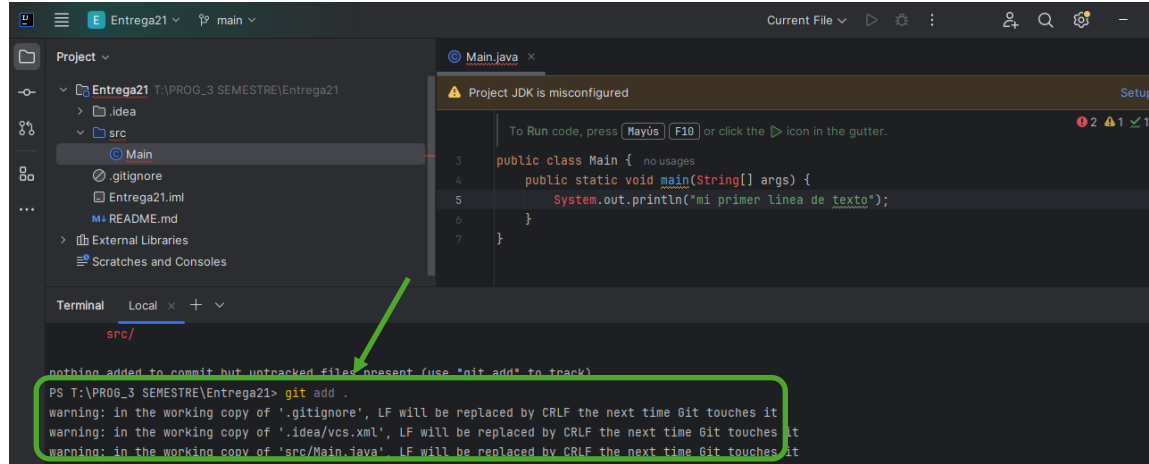
Una vez que se ha revisado los archivos modificados y se encuentran listos para confirmar, presionar la tecla "Enter".

Posteriormente se procede a usar el comando *git add* . para agregar el o los archivos que se requieran subir al repositorio.

Comando *git add* .

Cuando el código está listo y se da por terminado, se prepara una copia del estado del proyecto con el comando *git add* . , que agrega todos los cambios realizados al área de preparación. Esto permite que los cambios estén listos para ser confirmados por el comando *git commit* y finalmente subidos al repositorio remoto.(ver siguiente imagen).

Uso de git add . para preparar el archivo para el commit.



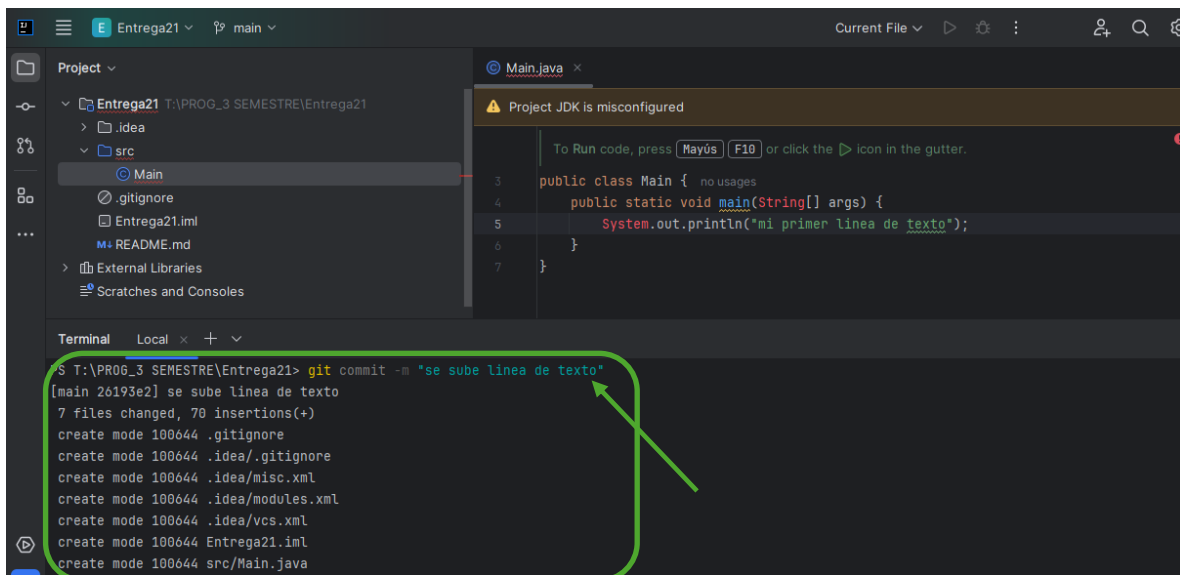
Nota. La imagen muestra cuando se ejecuta el comando *git add .*, Git agrega todos los archivos modificados y no rastreados al área de preparación. Posteriormente para ser confirmados con el *git commit -m ""*. Esto incluye archivos nuevos, archivos modificados y archivos eliminados. Fuente: Elaboración propia (2025).

Ahora que los archivos han sido añadidos al área de preparación, puedes proceder a hacer un *commit* con el comando *git commit -m "Tu mensaje de commit"*, guardando estos cambios en el historial de Git.

Comando *git commit -m "mensaje"*

El comando *git commit -m ""* crea una confirmación que es como una instantánea de tu repositorio en un momento específico. Cada confirmación representa el estado del proyecto en un punto determinado en el tiempo. Es importante que el mensaje sea claro y descriptivo, explicando brevemente qué cambios se hicieron. Esto ayuda a otros desarrolladores o al mismo autor a entender qué se modificó sin tener que revisar todo el código. (Ver siguiente imagen.)

Confirmación de cambios con git commit -m "Se sube línea de texto".



Nota. La imagen muestra que al ejecutar el comando `git commit -m "se sube linea de texto"`, se crea una confirmación que guarda los cambios realizados en el proyecto. En este caso, se agregan nuevos archivos y se realizaron modificaciones en el proyecto, como lo indican los detalles de la salida; Se modificaron 4 archivos en total, con 47 inserciones, y se crearon nuevos archivos: `.gitignore`, `.idea/.gitignore`, `.idea/vcs.xml` y `src/Main.java`.
Fuente: Elaboración propia (2025).

A continuación, se procede a subir los cambios que se realizaron “commits” a la rama remota que en este caso es la rama `main`. Con el comando `git push origin main`.

Comando `git push (rama)`

El comando `git push` se utiliza para enviar las confirmaciones realizadas en la rama local a un repositorio remoto. Este comando toma dos argumentos: un nombre remoto (por ejemplo, `origin`) y un nombre de rama (por ejemplo, `main`). Sin embargo, generalmente se ejecuta el comando `git push origin main` para subir los cambios locales a la rama `main` del repositorio remoto. (ver siguiente imagen).

Uso de `git push origin main` para subir cambios al repositorio remoto.

```

PS T:\PROG_3 SEMESTRE\Entrega21> git push origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 1.78 KiB | 304.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MonicaPoveda/Entrega21.git
 5133e34..26193e2  main -> main
PS T:\PROG_3 SEMESTRE\Entrega21>

```

Nota. La imagen muestra como el comando *git push origin main* se utiliza para enviar las confirmaciones locales de la rama main al repositorio remoto GitHub. Fuente: Elaboración propia (2025).

```

1  //TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
2  // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
3  public class Main {
4      public static void main(String[] args) {
5          System.out.println("mi primer línea de texto");
6      }
7  }

```

Nota. Al ejecutar este comando, los cambios realizados en el repositorio local se suben al repositorio en línea. Fuente: Elaboración propia (2025).

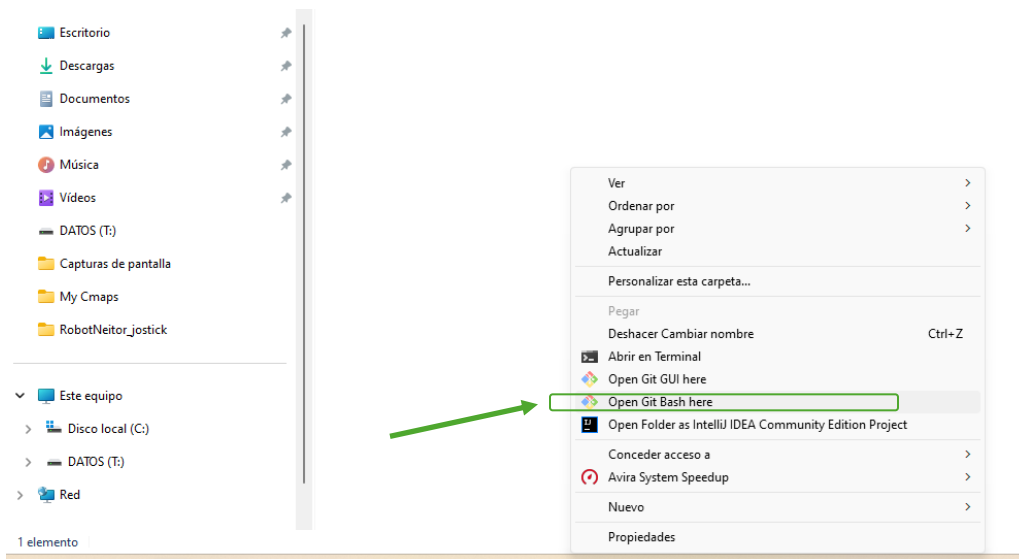
Comando *git clone*

El comando *git clone* se usa para crear una copia local de un repositorio remoto. Esto es útil cuando se desea trabajar en un proyecto que ya está alojado en un servidor de control de versiones, como GitHub.

Git descarga todo el historial del repositorio, incluyendo las ramas, los commits, y los archivos, permitiéndote trabajar en el proyecto como si fuera nuestro.

Tenemos el enlace del repositorio que queremos clonar.

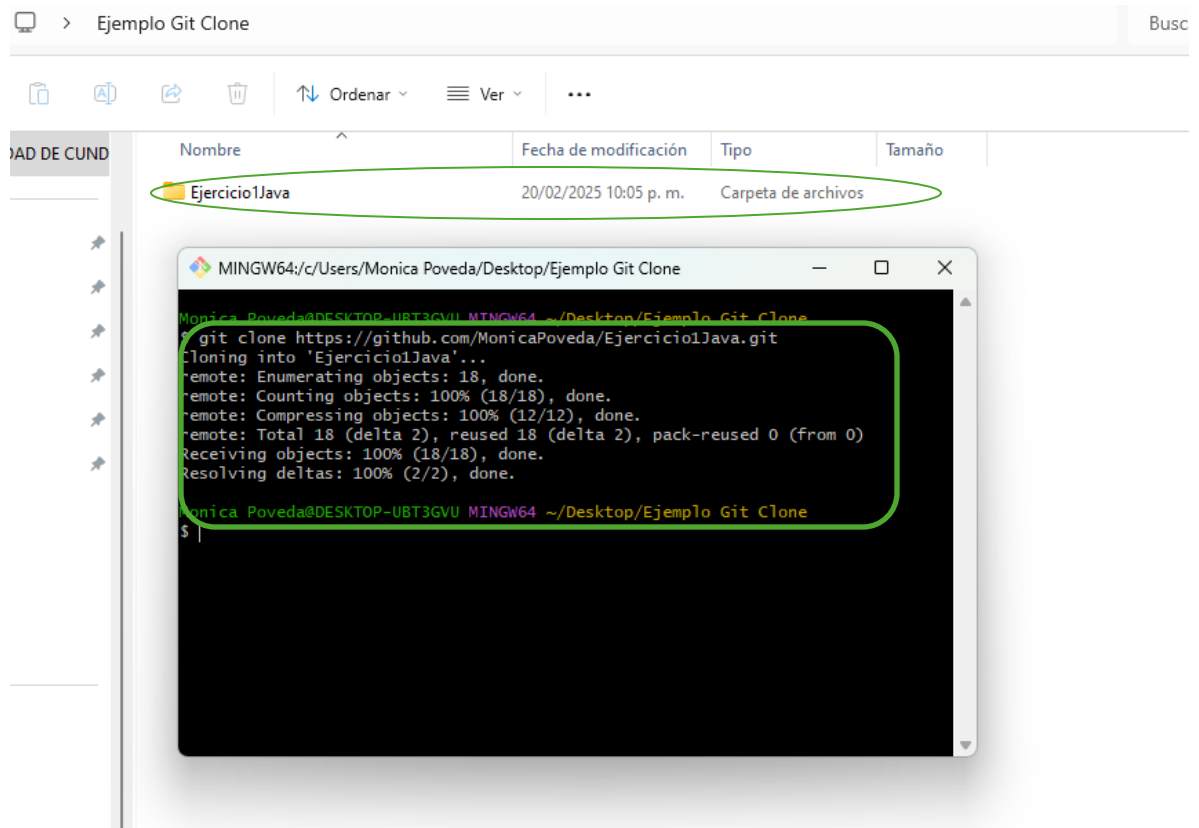
<https://github.com/Wamp-28/EjercicioJava1.git>



Nota. Abrimos una carpeta en nuestro ordenador y damos click derecho para abrir mas opciones y dar click en la copcion; *open get bash here*.

El *open get bash here* es simplemente una forma rápida de abrir una terminal Git en la ubicación exacta de una carpeta específica y dejar allí los archivos que se quiere clonar.

Al abrir esta terminal, introducimos el comando *git clone* seguido del link extensión *.git* . Posteriormente se clonará la carpeta que se encuentra dentro del repositorio a la carpeta que quisimos que quedara. Como se muestra en la siguiente imagen.



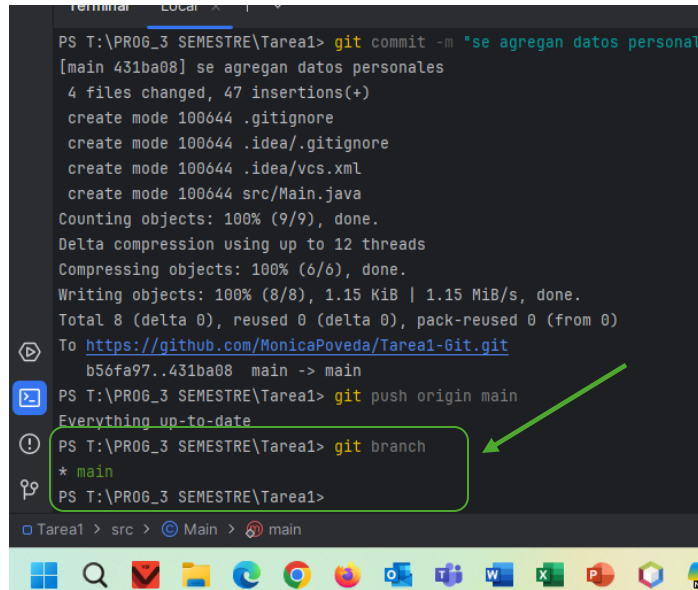
Nota. La imagen muestra como al introducir este comando automáticamente se clona la carpeta hacia nuestra carpeta base.

Una vez que tenemos el proyecto correctamente subido y confirmado en la rama principal, se crea por lo general otras ramas para trabajar en cambios específicos sin que se vea afectada directamente la rama principal. En este caso, después de tener el proyecto en la rama *main* se procede a saber en qué rama se está trabajando y posteriormente crear una nueva rama llamada *Rama2*.

Comando *git branch*

Para abrir una nueva rama, es esencial saber primero sobre que rama se está ejecutando. Para hacer esto, se usa el comando *git branch*, en la que muestra todas las ramas locales y resalta la rama en la que se está trabajando con un asterisco (*). (ver siguiente imagen)

Verificación de la Rama Actual con git branch.



```

PS T:\PROG_3 SEMESTRE\Tarea1> git commit -m "se agregan datos personales"
[main 431ba08] se agregan datos personales
 4 files changed, 47 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/vcs.xml
 create mode 100644 src/Main.java
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 1.15 KiB | 1.15 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MonicaPoveda/Tarea1-Git.git
 b56fa97..431ba08  main -> main
PS T:\PROG_3 SEMESTRE\Tarea1> git push origin main
Everything up-to-date
PS T:\PROG_3 SEMESTRE\Tarea1> git branch
* main
PS T:\PROG_3 SEMESTRE\Tarea1>

```

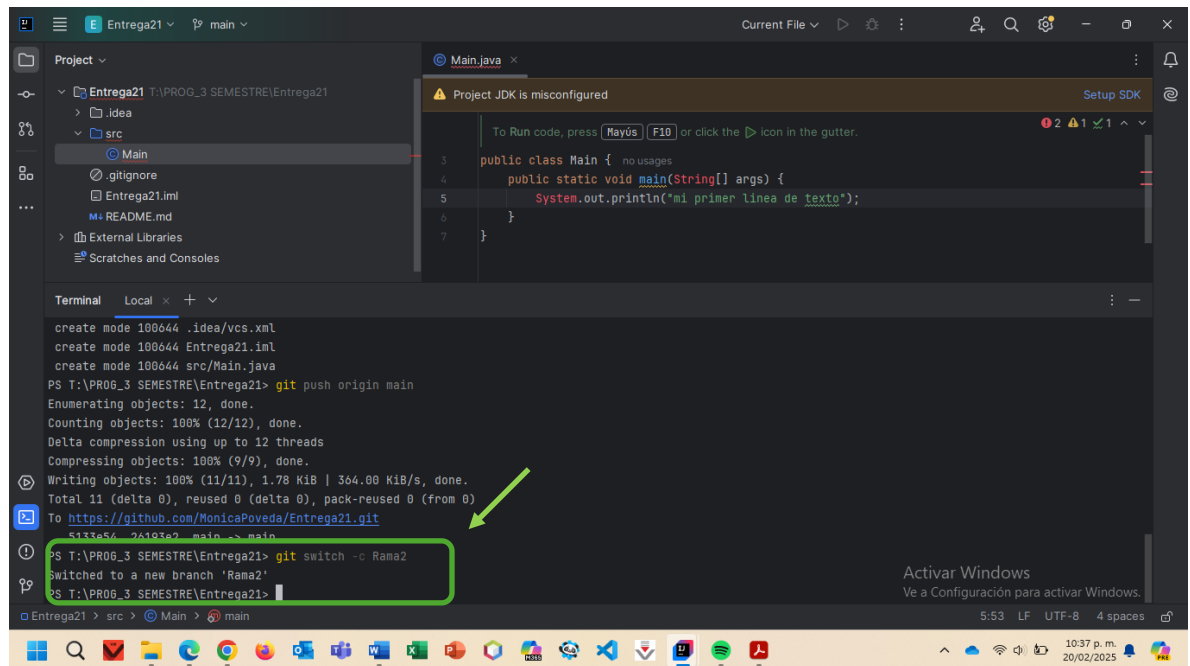
Nota. La imagen muestra como al ejecutar el comando *git branch* se despliegan las ramas locales del repositorio. En este caso, el comando muestra que se esta trabajando sobre la rama *main* , ya que esta resaltada y marcada con un asterisco (*). Fuente: Elaboración propia (2025).

Ahora, sabiendo que estamos sobre la rama *main*, y necesitamos crear una nueva rama lo ejecutamos con el comando *git switch -c Nombre de la rama*.

Comando *git switch -c* (nombre de la rama)

El comando *git switch -c* (nombre de la rama), se usa para crear y cambiar a una nueva rama, lo que permite empezar a realizar cambios sin afectar directamente a la rama principal (*main*). (Ver siguiente imagen).

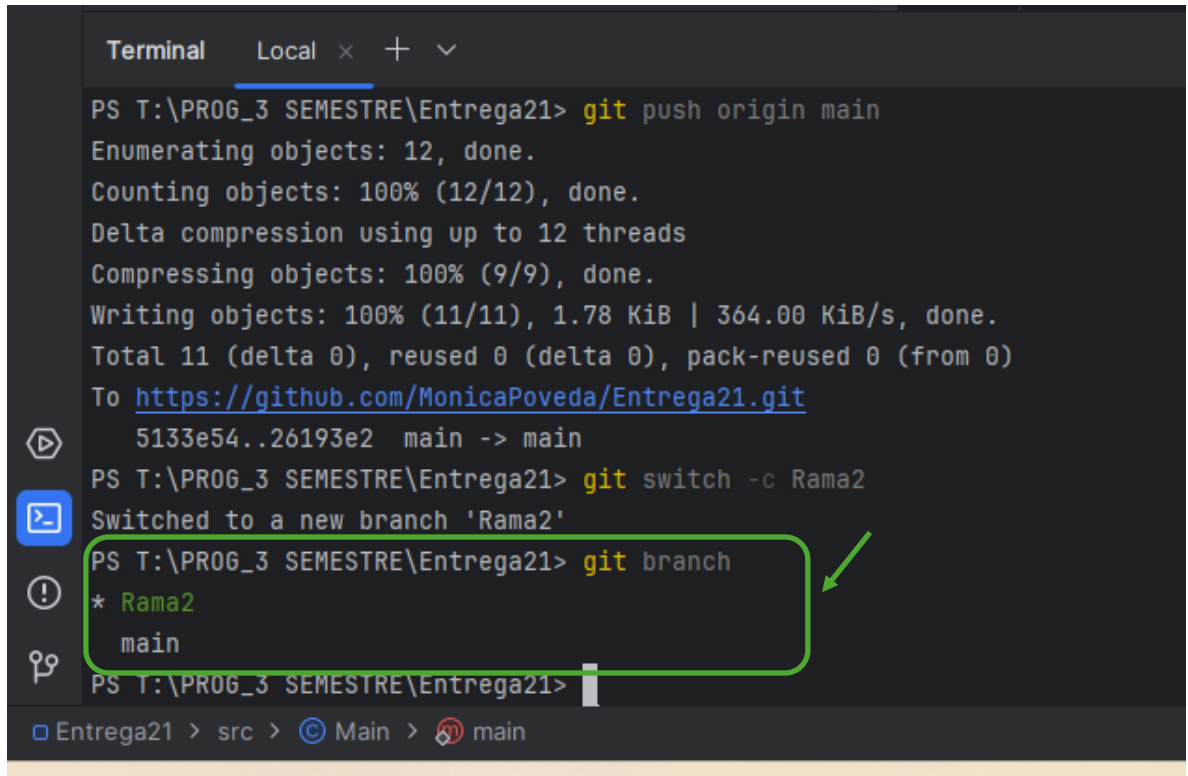
Creación de nueva rama “ Rama2”



Nota. En la imagen, se muestra como se utiliza el comando *git switch -c* (Nombre de la Rama) para crear una nueva rama llamada *Rama2*, que en este caso en la terminal confirma que la nueva rama ha sido creada y que el entorno de trabajo ha cambiado a la nueva rama. Fuente: Elaboración propia (2025).

Para verificar que el entorno de trabajo se ha cambiado a la nueva rama llamada *Rama2*, se ejecuta en la terminal el comando *git branch*, el cual ya se explicó anteriormente. Este comando muestra una lista de todas las ramas locales y marca con un asterisco (*) la rama en la que se encuentra actualmente. Si todo ha salido bien, se evidencia que la nueva rama *Rama2* está activa. (ver las siguientes imágenes).

Verificación de la Rama Activa con git switch -c (nombre de la rama)



```

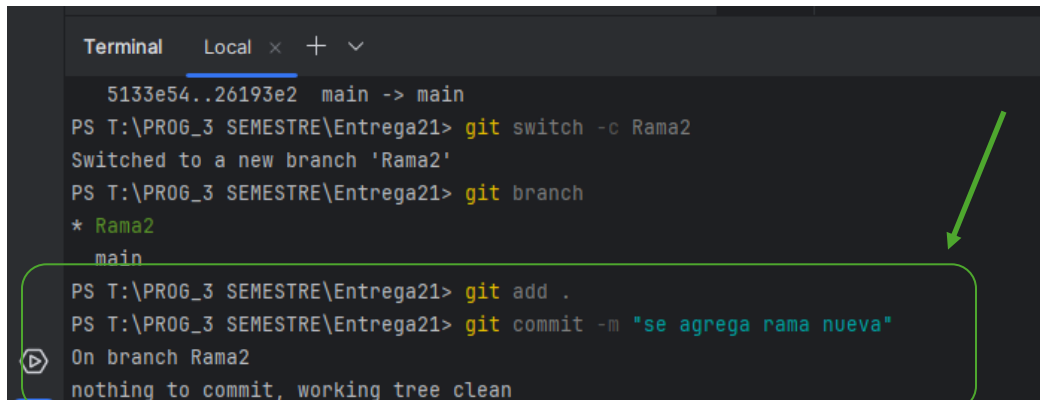
Terminal  Local x + v
PS T:\PROG_3 SEMESTRE\Entrega21> git push origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 1.78 KiB | 364.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/MonicaPoveda/Entrega21.git
5133e54..26193e2  main -> main
PS T:\PROG_3 SEMESTRE\Entrega21> git switch -c Rama2
Switched to a new branch 'Rama2'
PS T:\PROG_3 SEMESTRE\Entrega21> git branch
* Rama2
  main
PS T:\PROG_3 SEMESTRE\Entrega21>

```

Entrega21 > src > Main > main

Nota. La imagen muestra la salida del comando `git branch` ejecutado en la terminal, que permiten verificar que efectivamente se está trabajando sobre la nueva rama *Rama2*. Esto nos asegura que estamos listos para realizar modificaciones sin afectar la rama principal. Fuente: Elaboración propia (2025).

Estando ubicados en la nueva rama, procedemos a guardar los nuevos cambios antes de hacer un *commit*. Esto con el comando `add`. Después realizamos el *commit* el cual dice que se añade la nueva rama. como se muestra en la imagen.



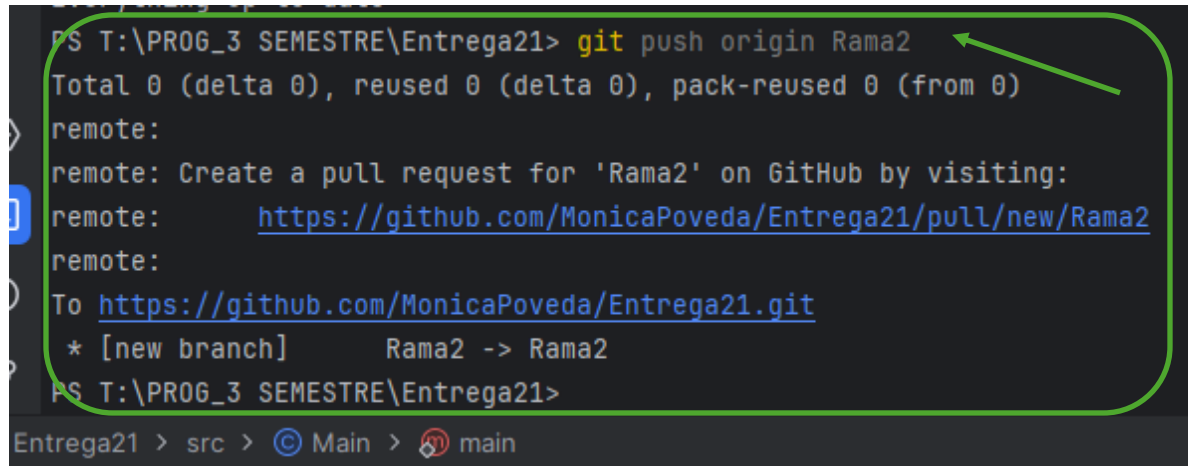
```

Terminal  Local x + v
5133e54..26193e2  main -> main
PS T:\PROG_3 SEMESTRE\Entrega21> git switch -c Rama2
Switched to a new branch 'Rama2'
PS T:\PROG_3 SEMESTRE\Entrega21> git branch
* Rama2
  main
PS T:\PROG_3 SEMESTRE\Entrega21> git add .
PS T:\PROG_3 SEMESTRE\Entrega21> git commit -m "se agrega rama nueva"
On branch Rama2
nothing to commit, working tree clean

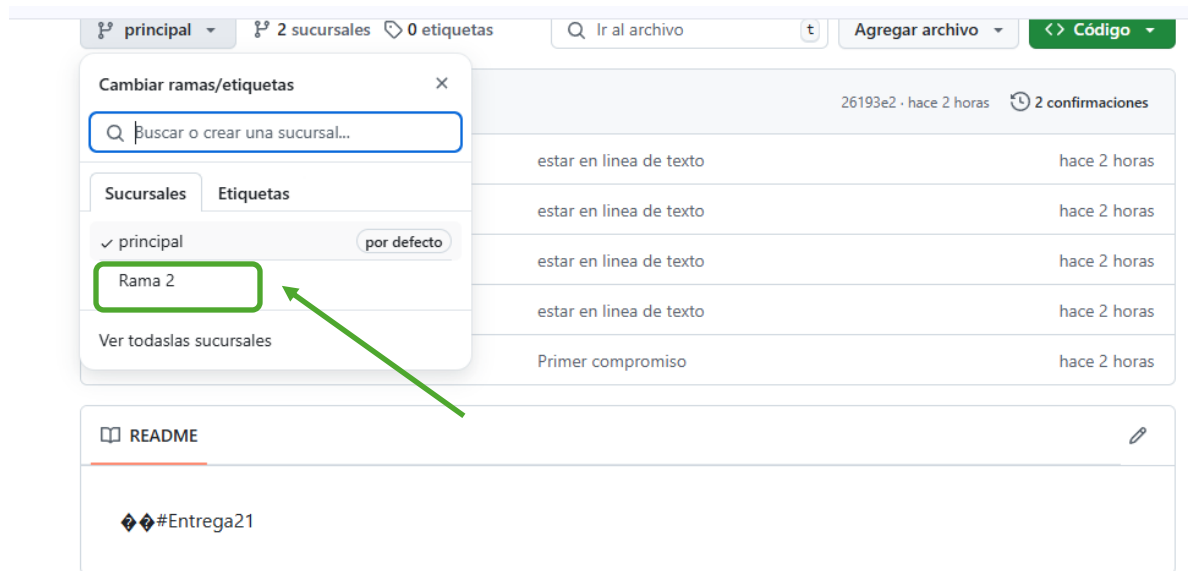
```

Se añaden los ajustes al código en la nueva rama y, posteriormente se publica en el repositorio remoto utilizando el comando `git push origin Rama2` (nombre de la rama).(ver la siguiente imagen).

Publicación de Cambios en la Rama Rama2



```
PS T:\PROG_3 SEMESTRE\Entrega21> git push origin Rama2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'Rama2' on GitHub by visiting:
remote:   https://github.com/MonicaPoveda/Entrega21/pull/new/Rama2
remote:
To https://github.com/MonicaPoveda/Entrega21.git
 * [new branch]      Rama2 -> Rama2
PS T:\PROG_3 SEMESTRE\Entrega21>
```

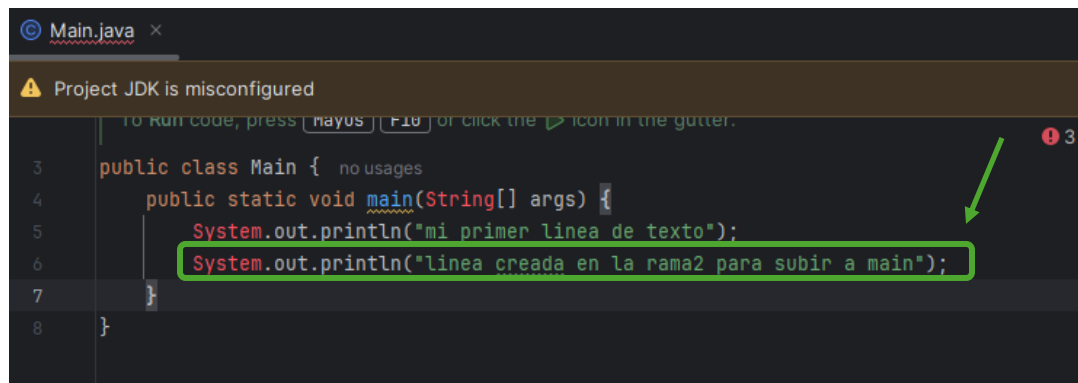



Nota. En la imagen1, se muestra el resultado del comando `git push origin Rama2` ejecutado en la terminal. Este comando sube los cambios realizados en la rama local *Rama2* al repositorio remoto en GitHub (imagen 2). El mensaje confirma que la rama ha sido subida correctamente y también proporciona un enlace para crear un (*pull request*) para

fusionar los cambios de la rama *Rama2* con la rama principal del repositorio. Fuente: Elaboración propia (2025).

Ahora para fusionar líneas de texto de las ramas creadas aparte de la rama *main* se realiza a través del comando *git pull origin (rama)*.

Entonces, como estamos en la rama *Rama2* vamos a agregar un alineo de texto para que se puede después subir cambios de la rama *Rama2* a la rama *main*.



```
1  Main.java x
2  Project JDK is misconfigured
3  To run code, press Mayus F10 or click the  icon in the gutter.
4  public class Main { no usages
5      public static void main(String[] args) {
6          System.out.println("mi primer linea de texto");
7          System.out.println("linea creada en la rama2 para subir a main");
8      }
9  }
```

Una vez añadido la línea de texto, se revisa que si se hayan registrado los cambios en la rama con un *git status*, que en este caso sería que se añadió una nueva línea de texto en el Código. Como se observa en la siguiente imagen.

```

T:\PROG_3 SEMESTRE\Entrega21
├── .idea
├── src
│   ├── Main
│   ├── .gitignore
│   ├── Entrega21.iml
│   └── README.md
├── External Libraries
└── Scratches and Consoles

Project JDK is misconfigured
To run code, press [mayus] [F10] or click the ▶ icon in the gutter.

3 public class Main { no usages
4     public static void main(String[] args) {
5         System.out.println("mi primer linea
6         System.out.println("linea creada en
7     }
8 }

Terminal Local x + v
PS T:\PROG_3 SEMESTRE\Entrega21> git push origin Rama2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'Rama2' on GitHub by visiting:
remote:   https://github.com/MonicaPoveda/Entrega21/pull/new/Rama2
remote:
To https://github.com/MonicaPoveda/Entrega21.git
* [new branch] Rama2 -> Rama2
PS T:\PROG_3 SEMESTRE\Entrega21> git status
On branch Rama2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/Main.java
  
```

Nota. La imagen muestra como efectivamente se realizo un cambio dentro de la carpeta de *src/main.java*, a lo que debemos aceptar esta modificación y subir los cambios con *git add* . y *git commit -m "se agrega línea de texto"* tal y como se muestra a continuación.

```

T:\PROG_3 SEMESTRE\Entrega21
├── .idea
├── src
│   ├── Main
│   ├── .gitignore
│   ├── Entrega21.iml
│   └── README.md
├── External Libraries
└── Scratches and Consoles

Project JDK is misconfigured
To run code, press [mayus] [F10] or click the ▶ icon in the gutter.

3 public class Main { no usages
4     public static void main(String[] args) {
5         System.out.println("mi primer linea de texto");
6         System.out.println("linea creada en la rama2 para subir a main");
7     }
8 }

Terminal Local x + v
(use "git restore <file>..." to discard changes in working directory)
        modified:   src/Main.java

no changes added to commit (use "git add" and/or "git commit -a")
PS T:\PROG_3 SEMESTRE\Entrega21> git add .
warning: in the working copy of 'src/Main.java', LF will be replaced by CRLF the next time Git touches it
PS T:\PROG_3 SEMESTRE\Entrega21> git commit -m "se agrega línea de texto a la rama2"
[Rama2 4523164] se agrega línea de texto a la rama2
1 file changed, 1 insertion(+)
  
```

Nota. La imagen muestra que efectivamente al guardar los cambios y agregar el *commit*, se ha insertado una nueva línea de texto.

Ahora para q este cambio se suba a la rama *Rama2* , debemos subir los cambios a la rama con el comando *git push origin Rama2*. Como se muestra en la siguiente imagen

```

4      public static void main(String[] args) {
5          System.out.println("mi primer linea de texto")
6          System.out.println("linea creada en la rama2 p
7      }
8  }

Terminal  Local x + v
no changes added to commit (use "git add" and/or "git commit -a")
PS T:\PROG_3 SEMESTRE\Entrega21> git add .
warning: in the working copy of 'src/Main.java', LF will be replaced by CRLF the next time Git touches it
PS T:\PROG_3 SEMESTRE\Entrega21> git commit -m "se agrega linea de texto a la rama2"
[Rama2 4523164] se agrega linea de texto a la rama2
1 file changed, 1 insertion(+)
PS T:\PROG_3 SEMESTRE\Entrega21> git push origin Rama2
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 424 bytes | 424.00 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/MonicaPoveda/Entrega21.git

```

Nota. En la imagen se muestra como al ejecutar el comando *git push origin Rama2* se suben exitosamente los cambios que se realizaron.

Este cambio se realizo sobre la rama actual, eso quiere decir que la rama principal no tiene este cambio. Para que se bajen los cambios que se hicieron en la rama *Rama2* a la rama *main* se debe ubicar en la rama que se desea que aparezcan cambios y posteriormente bajar los cambios mediante un *git pull origin Rama2*.

Comando *git pull origin (rama)*

El comando *git pull origin (rama)* se emplea para extraer y descargar contenido desde un repositorio remoto y actualizar al instante el repositorio local para reflejar ese contenido.

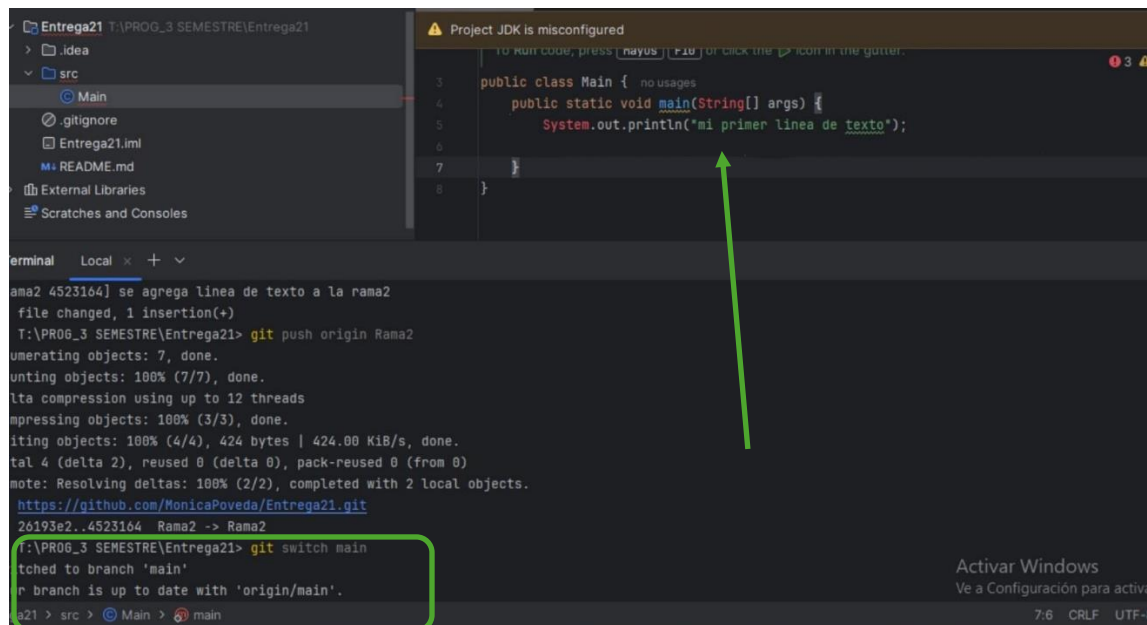
Como queremos que se bajen los cambios a la rama *main* entonces nos devolvemos a ubicarnos a la rama *main*. Este cambio de rama se realiza a través del comando *git switch (rama)*.

Comando *git switch (rama)*

El comando *git switch (rama)* permite cambiar la rama HEAD actual.

Este comando es relativamente nuevo (se agregó en Git v2. 23) y ofrece una alternativa más simple al comando clásico "checkout". Antes de que "switch" estuviera disponible, el cambio de ramas se debía realizar con el comando "checkout".

Por consiguiente, realizamos el cambio de rama con el comando *git switch main* y como ya nos ubicaremos en la rama en la que se quiere bajar los cambios de *rama2* , entonces después ejecutamos el comando de *git pull origin Rama2*, esto con el fin de q aparezca la línea q introducimos en *Rama2* a la rama *main*. Así como se muestra a continuación.



The screenshot shows an IDE with a project named 'Entrega21'. The file explorer on the left shows the project structure. The main editor displays a Java file 'Main.java' with the following code:

```

3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("mi primer línea de texto");
6     }
7 }

```

The terminal window at the bottom shows the following commands and output:

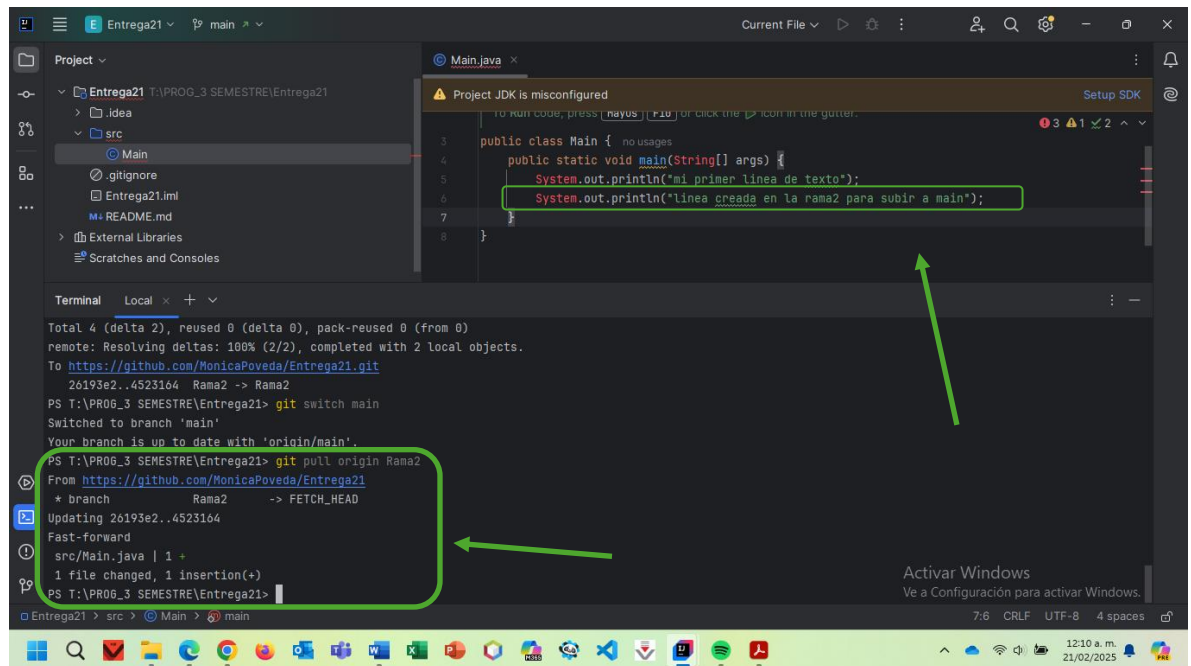
```

ama2 4523164] se agrega línea de texto a la rama2
file changed, 1 insertion(+)
T:\PROG_3 SEMESTRE\Entrega21> git push origin Rama2
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 424 bytes | 424.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
Note: Resolving deltas: 100% (2/2), completed with 2 local objects.
https://github.com/MonicaPoveda/Entrega21.git
26193e2..4523164 Rama2 -> Rama2
T:\PROG_3 SEMESTRE\Entrega21> git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
Entrega21 > src > Main > main

```

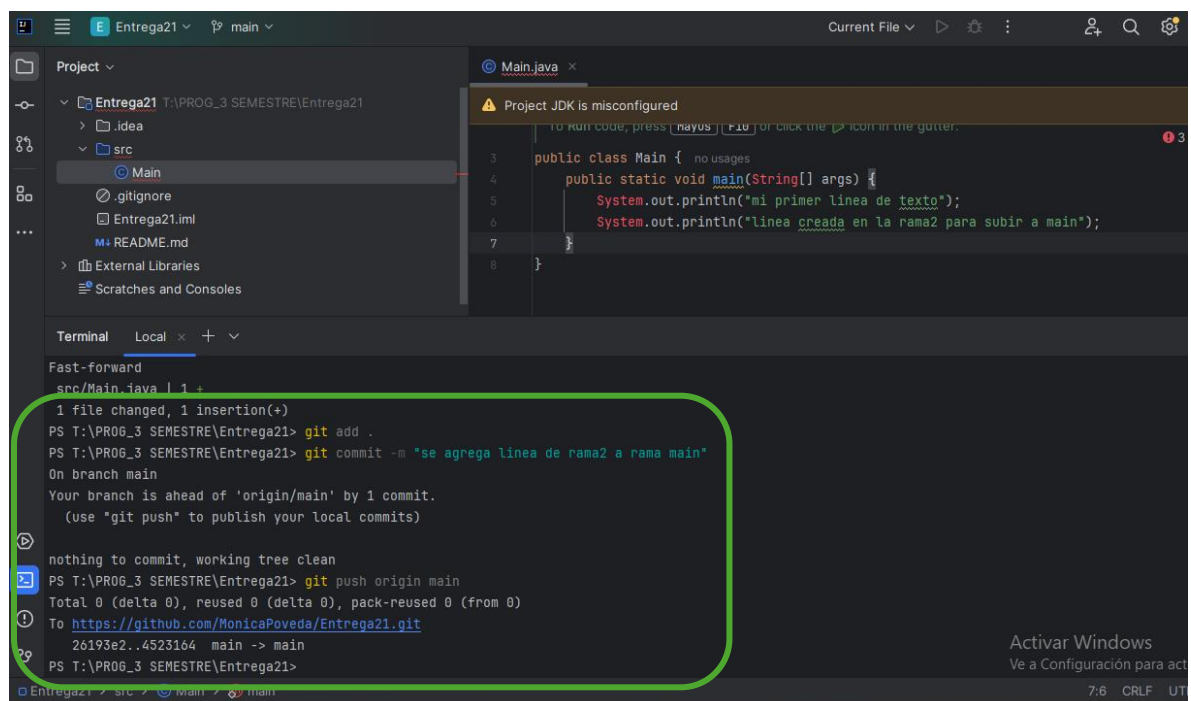
A green box highlights the `git switch main` command and its output in the terminal. A green arrow points from the terminal to the code editor, indicating the change in the code.

Nota. Aquí podemos observar cómo al pasarnos a la rama *main* no aparece la línea de texto “línea creada en la Rama2 para subir a main” modificada y publicada en la Rama2. Entonces para q nos aparezca este cambio ejecutamos ahora el *git pull Rama2*.



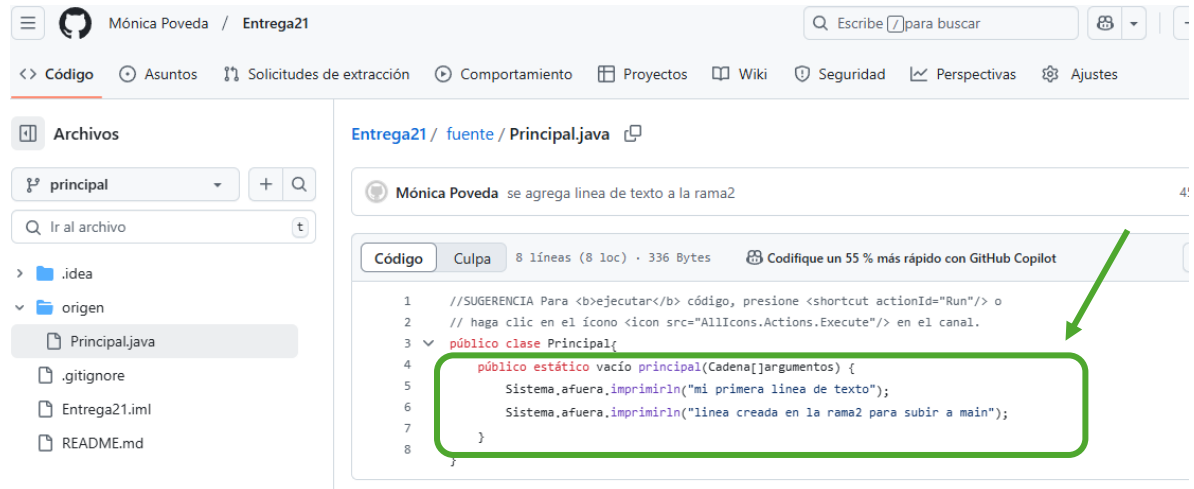
Nota. la imagen muestra como al introducir `git pull origin main` bajalos cambios a la rama *main*.

Ahora, debemos subir cambios a la rama *main*. Esto con los comandos `git add .` ; `git commit -m " "` y subir los cambios a la rama con `git push origin main`. Como se muestra a continuación.



Nota. La imagen muestra como con `git add . ; git commit -m " "` se guardan los cambios realizados sobre la rama `main` y con `git push origin main` para subir los cambios a la rama `main`.

Ahora verificamos en el repositorio si se realizaron los cambio.



Nota. Efectivamente en el repositorio si nos dirigimos a `src – main`. Podemos ver como nuestro comentario que se creo desde la rama `Rama2` se fusiono correctamente con la rama `main`.

Por otra parte, cuando ya no se necesita una rama diferente a la `main` se puede borrar mediante el comando `git switch -D (rama a eliminar)`

Comando `git branch -D (rama a eliminar)`.

El comando `git branch -D (rama a eliminar)`, se usa para eliminar una rama de forma forzada por ello la “D” en mayúscula, incluso si tiene cambios no confirmados. Este comando es útil cuando se desea eliminar una rama que ya no necesitas y se quiere asegurar de que sea eliminada independientemente de si tiene o no cambios pendientes. Para asegurarnos que se ha eliminado correctamente nuestra rama, podemos verificar con el comando `git branch`, que nos muestra en que rama estamos trabajando y cuales hay. Como se muestra a continuación.


```

remote: Create a pull request for 'RamaElim' on GitHub by visiting:
remote: https://github.com/MonicaPoveda/Entrega21/pull/new/RamaElim
remote:
To https://github.com/MonicaPoveda/Entrega21.git
 * [new branch]      RamaElim -> RamaElim
PS T:\PROG_3 SEMESTRE\Entrega21> git branch
Rama2
 * RamaElim
main
PS T:\PROG_3 SEMESTRE\Entrega21>

```

Nota. En la imagen se puede observar como al ejecutar el *git branch* se despliegan todas las ramas con las que se están trabajando.

Ahora bien, si se desea eliminar la rama *RamaElim*, entonces introducimos el *git* que se nombraba anteriormente; *git branch -D RamaElim*. Para así, poder eliminar la rama específicamente *RamaElim*. Seguido de esto se vuelve a cargar los cambio y a subirlo ala rama main; *git add . git commit -m ""* y por ultimo el *git push origin main*.

```

> External Libraries
Scratches and Consoles

Terminal Local x + v
PS T:\PROG_3 SEMESTRE\Entrega21> git branch
Rama2
RamaElim
* main
PS T:\PROG_3 SEMESTRE\Entrega21> git branch -D RamaElim
Deleted branch RamaElim (was 4523164).
PS T:\PROG_3 SEMESTRE\Entrega21> git add .
PS T:\PROG_3 SEMESTRE\Entrega21> git commit -m "se elimina la rama RamaElim"
On branch main
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
PS T:\PROG_3 SEMESTRE\Entrega21> git push origin main
Everything up-to-date

```

Nota. en la imagen se muestra como introduciendo el *git branch -D RamaElim* y posteriormente cargar los cambios y subirlo a la rama main, dice que fue exitoso la eliminación de la rama. Para verificar si se eliminó correctamente lo ejecutamos desde el *git branch*.

```

src
├── Main
├── .gitignore
├── Entrega21.iml
├── README.md
├── External Libraries
└── Scratches and Consoles

3 public class Main { no usages
4     public static void main(String[] args) {
5         System.out.println("mi primer linea de texto");
6         System.out.println("linea creada en la rama2 para su
7     }
8 }

Terminal Local (2)
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

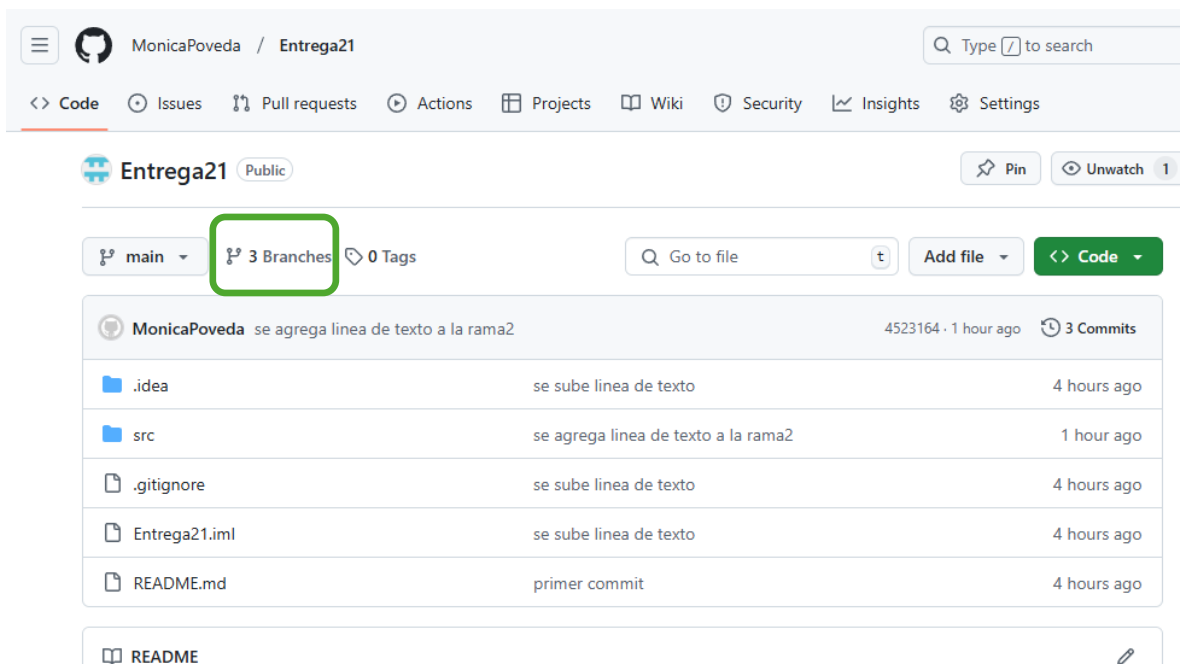
PS T:\PROG_3 SEMESTRE\Entrega21> git branch
Rama2
* main
PS T:\PROG_3 SEMESTRE\Entrega21>

```

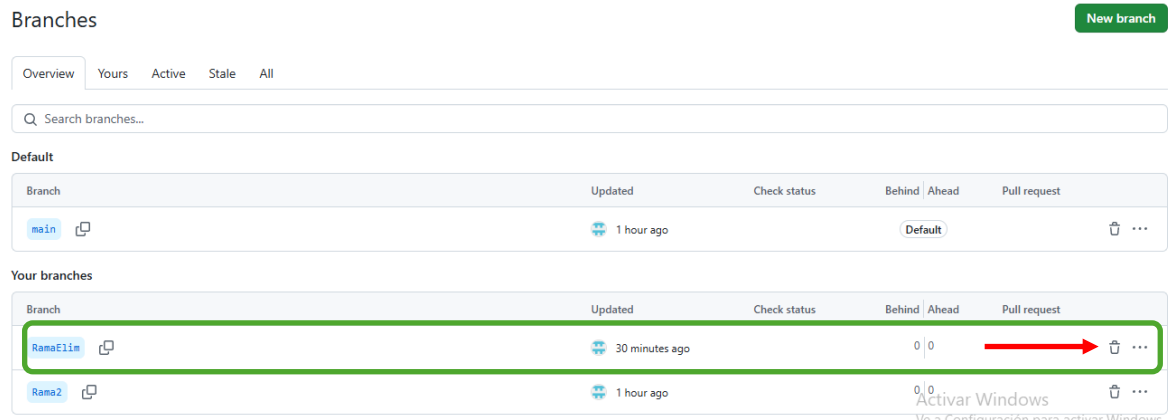
Nota. En la imagen se puede observar como efectivamente al hacer *git branch* este nos muestra que se elimino la rama local *RamaElim*.

Recordemos que cuando se borra una rama local también se debe borrar en la rama remota que está ubicada en github.

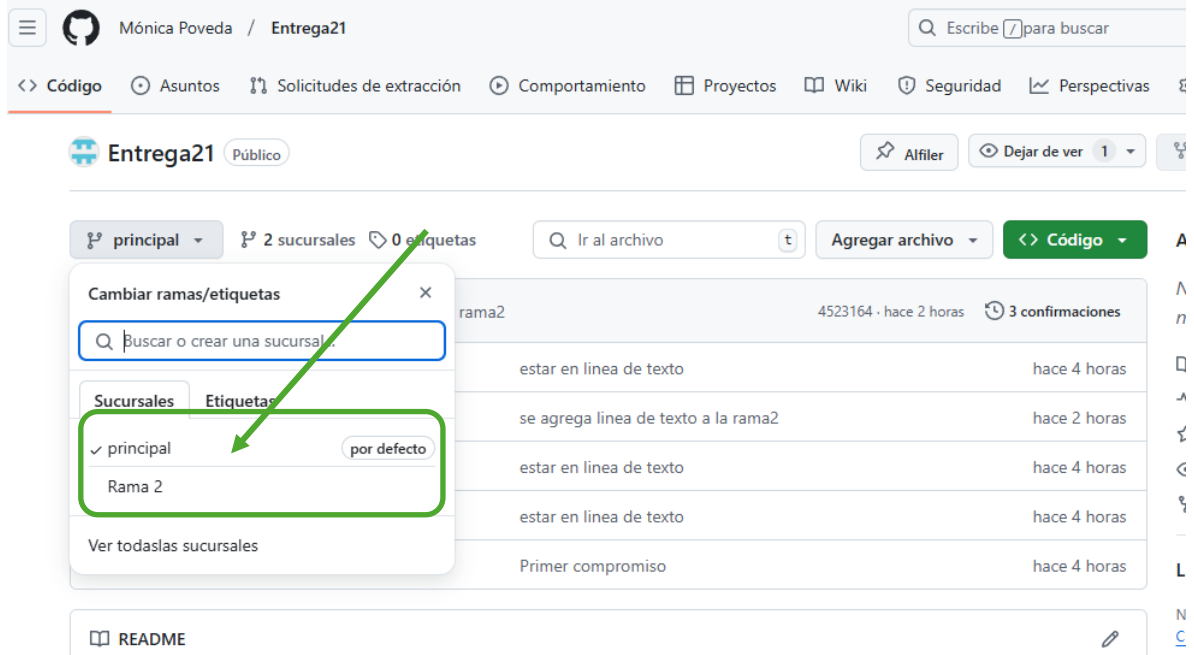
Para eliminar la rama en remoto, debemos dirigirnos al repositorio y ubicar la opción “Branches” y posteriormente dar click sobre el.



Una vez entramos, nos dirigiremos a la rama que borramos en el local. En este caso *RamaElim*. y nos ubicaremos en la figura de papelera “Delete branch” y damos click.



Una vez eliminada la rama, nos dirigimos al panel de las ramas, por lo cual no va aparecer ya que se ha eliminado efectivamente.



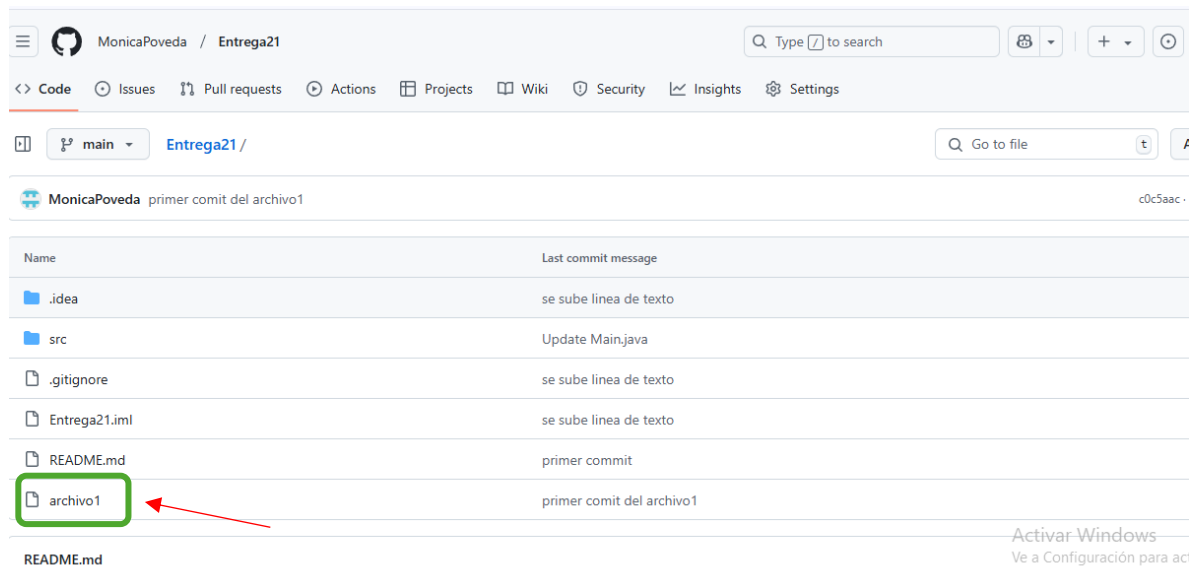
Comando *fetch* -all

Es básicamente un movimiento de poder; el *git fetch* actualiza las copias locales de las ramas remotas, por lo que siempre es seguro para sus ramas locales, PERO:

- ✓ *Git fetch* no actualizará las ramas locales (que rastrean las ramas remotas); Si desea actualizar sus ramas locales, aún debe extraer todas las ramas.
- ✓ *Git fetch* no creará ramas locales (que rastrean ramas remotas), debe hacerlo manualmente. Si desea enumerar todas las sucursales remotas: `git branch -a`.

La diferencia entre el *git pull* al *git fetch* es que *pull* descarga la información, pero este va hacer *git merge* que prácticamente es unir las ramas a nuestra rama principal *main* de nuestro repositorio local.

Entonces para iniciar con el comando *git fetch* es porque alguna persona que está colaborando en la rama *main* realizo un cambio. Como se observa en la siguiente imagen donde nuestro colaborador añadir el *archivo1*.



Entonces si yo me dirijo al repositorio, este archivo no va a aparecer, entonces procedemos a abrir la terminal y escribimos *git fetch -all*. Ver la siguiente imagen)

```

Terminal Local x Local (2) x Local (3) x Local (4) x Local (5) x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS T:\PROG_3 SEMESTRE\Entrega21> git fetch --all
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 1004 bytes | 55.00 KiB/s, done.
From https://github.com/MonicaPoveda/Entrega21
  a4e0058..c0c5aac main    -> origin/main
PS T:\PROG_3 SEMESTRE\Entrega21>
Entrega21 > src > © Main


```

Entonces para generar el *git fetch --all* en la terminal, este nos va a indicar que ya se descargó la información desde el repositorio remoto y que en este momento esta en nuestra rama principal.

Por lo que cada vez que introducimos un *git fetch* este crea una rama oculta que no estamos viendo.

Comando *branch -r*

Se utiliza para listar las ramas remotas del repositorio en Git. Específicamente, muestra todas las ramas que existen en los remotos configurados, como origin (el nombre predeterminado del remoto). Como se muestra a continuación.



```
PS T:\PROG_3 SEMESTRE\Entrega21> git branch -r
origin/HEAD -> origin/main
origin/Rama2
origin/RamaBranch
origin/RamaElim
origin/main

PS T:\PROG_3 SEMESTRE\Entrega21> git push origin --delete RamaBranch
To https://github.com/MonicaPoveda/Entrega21.git
- [deleted]          RamaBranch

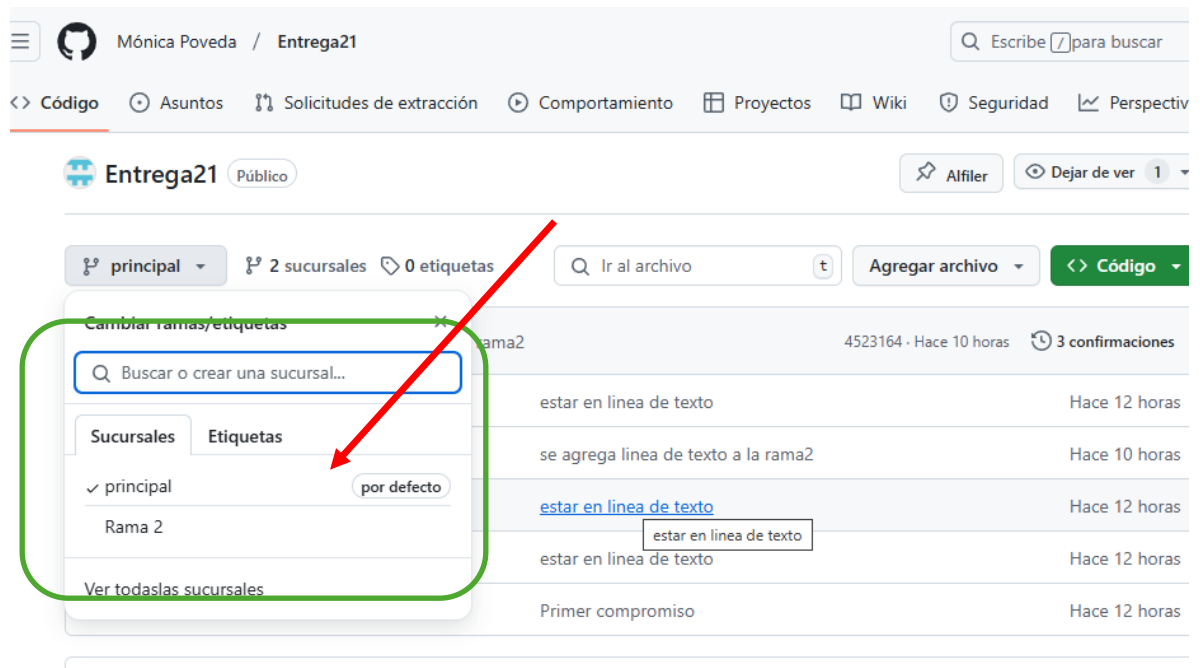
PS T:\PROG_3 SEMESTRE\Entrega21> git branch -r
origin/HEAD -> origin/main
origin/Rama2
origin/RamaElim
origin/main

PS T:\PROG_3 SEMESTRE\Entrega21>
```

Nota. la imagen muestra como al introducir el comando *git branch -r* este despliega todas las ramas tanto locales como remotas.

Seguido de estos se desea eliminar una rama remota, para eliminarla debemos utilizar el comando *git push origin --delete (rama remota a eliminar)*. Por lo que al volver a ejecutar el comando *git branch -r* ya no mostrara la rama remota *RamaBranch*.

Y para verificar que se ha eliminado la rama remota, nos dirigimos al repositorio y esta ya no aparecerá, tal cual como se muestra continuación.



Otra ocasión que sucede es cuando por accidente se elimina un commit, lo podemos deshacer con un comando llamado *git revert HEAD* que en este caso seria para revertir el ultimo comit o *git revert (# commit)* para revertir un commit especifico.

Comando *git revert*

Este comando es una forma de “deshacer” un commit de forma segura y confiable de tal modo que no afecta el trabajo con otros colaboradores.

Para revertir algún commit especifico, busca primero el ID del commit el cual quiere revertir, esto a través de formas: *git log* , *git reflog* o *git --oneline*. O simplemente revertir el ultimo commit con *HEAD*.

Comando *git log*

El comando *git log* muestra el historial completo de los commits en la rama actual (o en cualquier otra rama especifica) Por defecto, se muestra una lista detallada de commits, incluyendo:

- Hash del commit (identificador único).
- Autor del commit.
- Fecha y hora en que se realizó el commit.
- Mensaje de commit

Tal y como se muestra a continuación.

```

no changes added to commit (use "git add" and/or "git commit -a")
PS T:\PROG 3 SEMESTRE\Entrega21> git log
commit 4523164eb118fa92fafdd195c2837e3b5eb8ffee (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main)
Author: MonicaPoveda <monicapoveda337@gmail.com>
Date: Thu Feb 20 23:16:59 2025 -0500

    se agrega linea de texto a la rama2

commit 26193e2324cf270ed3c199d1ed380741aaf5a761
Author: MonicaYulihet <monicapoveda337@gmail.com>
Date: Thu Feb 20 20:53:33 2025 -0500

    se sube linea de texto

commit 5133e54c9d28b622f3b9201cc659a149ebc802
Author: MonicaYulihet <monicapoveda337@gmail.com>
Date: Thu Feb 20 20:48:55 2025 -0500
  
```

Comando *git reflog*

El comando *git reflog* muestra el historial de los movimientos de HEAD (el puntero que indica en qué commit o rama en la que se encuentre en ese momento). A diferencia de *git log*, que solo muestra los commits, *git reflog* muestra todas las acciones realizadas, como cambios de ramas, pulls, checkouts, merges, entre otros. (ver la siguiente imagen)

```

PS T:\PROG 3 SEMESTRE\Entrega21> git reflog
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{0}: checkout: moving from RamaBranch to Rama2
f69343f (RamaBranch) HEAD@{1}: commit: se crea la rama RamaBranch para ser eliminada con --delete
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{2}: checkout: moving from Rama2 to RamaBranch
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{3}: checkout: moving from main to Rama2
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{4}: checkout: moving from RamaElim to main
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{5}: checkout: moving from main to RamaElim
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{6}: pull origin Rama2: Fast-forward
26193e2 HEAD@{7}: checkout: moving from Rama2 to main
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{8}: commit: se agrega linea de texto a la rama2
26193e2 HEAD@{9}: checkout: moving from main to Rama2
26193e2 HEAD@{10}: commit: se sube linea de texto
5133e54 HEAD@{11}: Branch: renamed refs/heads/master to refs/heads/main
5133e54 HEAD@{13}: commit (initial): primer commit
  
```

Comando *log --oneline*

El comando *git log --oneline* es una versión más compacta y resumida del *git log*. Muestra el historial de los commits pero con un solo resumen por commit, donde se muestra solo el hash corto (primeros 7 caracteres del hash) y el mensaje del commit.


```

4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{4}: checkout: moving from RamaElim to main
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{5}: checkout: moving from main to RamaElim
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{6}: pull origin Rama2: Fast-forward
26193e2 HEAD@{7}: checkout: moving from Rama2 to main
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) HEAD@{8}: commit: se agrega linea de texto a la rama2
26193e2 HEAD@{9}: checkout: moving from main to Rama2
26193e2 HEAD@{10}: commit: se sube linea de texto
5133e54 HEAD@{11}: Branch: renamed refs/heads/master to refs/heads/main
5133e54 HEAD@{13}: commit (initial): primer commit
PS T:\PROG_3 SEMESTRE\Entrega21> git log --oneline
4523164 (HEAD -> Rama2, origin/main, origin/RamaElim, origin/HEAD, main) se agrega linea de texto a la rama2
26193e2 se sube linea de texto
5133e54 primer commit

```

Entonces como queremos revertir el ultimo commit lo podemos hacer por *HEAD* o por el commit específico.

En este caso introducimos el comando *HEAD*, después de presionar la tecla enter se nos despliega en la terminal la opción que permite cambiar el nombre del commit el cual se quiso revertir; esto con la tecla (a). Si no, se presiona la tecla *Esc* para salir de esa configuración y se procede a escribir *:wq* para confirmar los cambios. Como se muestra a continuación.

```

Terminal Local x + v
Revert "Guardar cambios antes del revert"
This reverts commit fc79b4064983ed10ace5d083d3d95a658498aa0c.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is ahead of 'origin/main' by 1 commit.
# (use "git push" to publish your local commits)
#
git/COMMIT_EDITMSG [unix] (10:33 20/02/2025)
:wq

```

Después de esto, aparecerá que se revirtió una línea de texto que posteriormente se había eliminado, si vemos en la parte superior derecha del código, ya aparecerá esta línea de texto. Como se muestra a continuación.

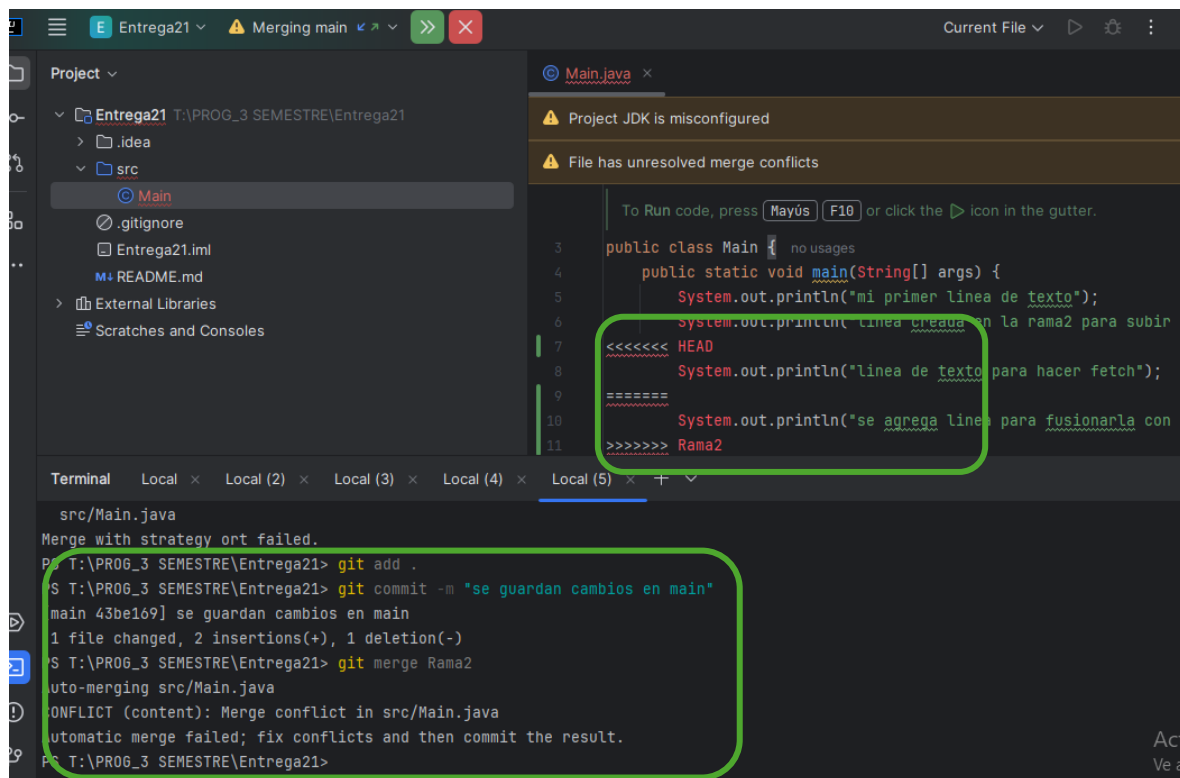
The screenshot shows an IDE with a project named 'Entrega21'. The 'Main.java' file is open, showing a Java class with a `main` method. A red arrow points from the terminal window to the line `System.out.println("línea creada en la rama2 para subir a main");` in the code. The terminal window shows the command `git revert HEAD` being executed, which reverts the last commit. The output of the command is: `Revert "se guardan cambio antes de revertir"`, `1 file changed, 1 insertion(+)`, and `T:\PROG_3 SEMESTRE\Entrega21>`.

Nota. se evidencia como al introducir el comando con HEAD este nos arroja que se ha revertido el ultimo commit.

Comando *git merge*

Inicialmente el comando *git merge* se utiliza para combinar los cambios de una rama en otra. Cuando se ejecuta *git merge*, Git genera un nuevo commit de fusión si las ramas tienen cambios divergentes.

Para iniciar con el merge primero guardamos cambio con *git add* . y *git commit*. Para que permita hacer el *git merge*.



la imagen muestra como efectivamente se hace un merge pero se encuentran conflictos dentro de los archivos, para solucionar esto, eliminamos los ismbolos de == y >>>> en las líneas que aprezcan , después damos click en la parte superior “resolver conflicto”.



Una vez realizado el merge, aparecerá los cambios en la línea de Código como se muestra a continuación:

```

To Run code, press Mayús F10 or click the ▶ icon in the gutter.

3 ▶ public class Main {
4 ▶     public static void main(String[] args) {
5         System.out.println("mi primer linea de texto");
6         System.out.println("linea creada en la rama2 para subir a main");
7         System.out.println("linea de texto para hacer fetch");
8     }
9 }

Local (2) x Local (3) x Local (4) x Local (5) x + v
Entrega21> git merge --abort
Entrega21> git branch

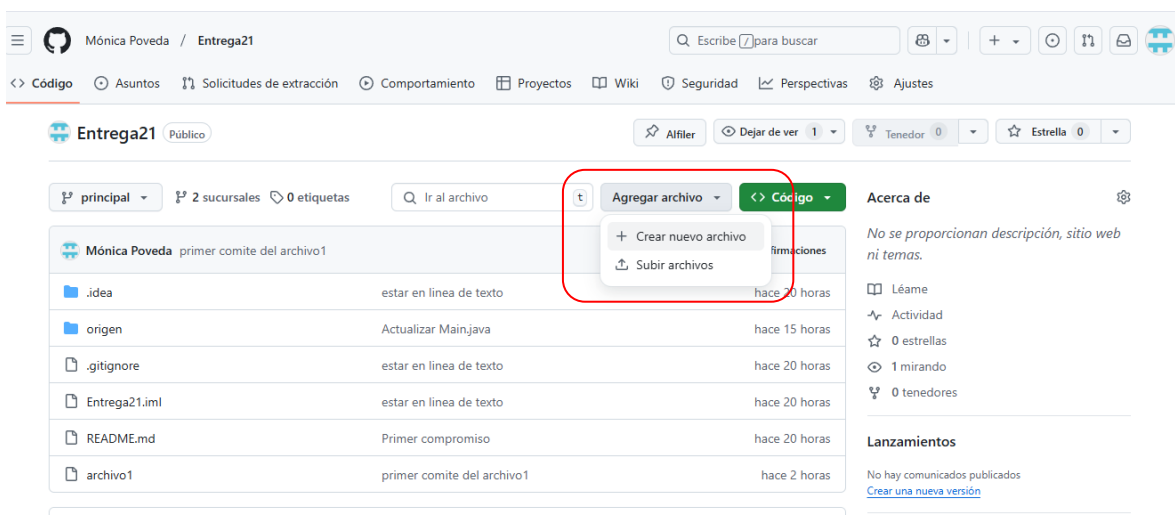
```

Git rebase

Recopila uno a uno los cambios confirmados en una rama, y reaplicarlos sobre otra. Utilizar rebase nos puede ayudar a evitar conflictos siempre que se aplique sobre commits que están en local y no han sido subidos a ningún repositorio remoto.

¿Como asociar una carpeta con un repositorio existente?

para agregar un archivo a un repositorio nos dirigimos al repositorio en la opción agregar archivo como se muestra a continuación y se sube:



Por último se incorpora el link del repositorio donde fue cargado el código.

<https://github.com/MonicaPoveda/Entrega21.git>

Conclusión

En conclusión, se evidencia el proceso de integración de un proyecto con un repositorio en GitHub el cual no solo facilita la sincronización entre un repositorio local y remoto, sino que también optimiza el trabajo en equipo al permitir que varios desarrolladores colaboren en un mismo proyecto sin interferir con el trabajo de los demás. Asimismo, comandos como *git switch -c* para crear nuevas ramas y *git branch -D* para eliminar ramas que son cruciales para mantener la organización en proyectos más grandes, permitiendo que cada cambio se trabaje de manera aislada.

Referencias bibliográficas

Lopez-Pellicer, F. J., Béjar, R., Latre, M. A., Noguera-Iso, J., & Zarazaga-Soria, F. J. (2015, 8 julio). *GitHub como herramienta docente*.

<https://upcommons.upc.edu/handle/2117/76761>

Atlassian. (s. f.). git init | Atlassian Git Tutorial.

<https://www.atlassian.com/es/git/tutorials/setting-up-a-repository/git-init>

git Guides. (2025). GitHub. <https://github.com/git-guides>

Programmer Exception. (2020, 27 septiembre). *11 Cómo usar git pull y cómo traer los últimos cambios de un Repositorio Remoto de GitHub* [Video].

YouTube. <https://www.youtube.com/watch?v=aTlJ7o-xFk4>

Astigarraga, J., & Cruz-Alonso, V. (2022). ¿ Se puede entender cómo funcionan Git y GitHub!. *Ecosistemas*, 31(1), 2332-2332.

<https://www.revistaecosistemas.net/index.php/ecosistemas/article/download/2332/1505>

Yanguas, S. T. (2021, 14 abril). *Explicación del comando Git Log*. freeCodeCamp.org. <https://www.freecodecamp.org/espanol/news/explicacion-del-comando-git-log/>

Agregar un archivo a un repositorio - Documentación de GitHub. (s. f.).

GitHub Docs. <https://docs.github.com/es/repositories/working-with-files/managing-files/adding-a-file-to-a-repository>