



UNIVERSIDAD DE LA FRONTERA

FACULTAD DE INGENIERÍA Y CIENCIAS

Departamento de Ciencias de la Computación e Informática

RT-POO-02 Uso de GitHub

Profesor

Dr. Samuel Sepúlveda Cuevas

Ayudante

Jesús Tapia Martin

Código

ICC490

Temuco, Chile.

Abril, 2025

Índice

1	Introducción	3
2	Manejo de los cambios en un repositorio	4
2.1	Commit - Registrar cambios locales	4
2.1.1	«Commit» según Pro Git	4
2.1.2	Realizar Commit	4
2.1.3	Revisar cambios - Diff	5
2.1.4	Commit Message	6
2.1.5	Amend - Modificar el último Commit	7
2.2	Push - Subir cambios al remoto	8
2.2.1	«Push» según Pro Git	8
2.2.2	Realizar Push	8
2.3	Pull - Traer cambios del remoto	10
2.3.1	«Pull» según Pro Git	10
2.3.1	Realizar «Pull»	10
3	Trabajo con ramas	12
3.1	¿Qué es una rama?	12
3.2	Crear rama	12
3.2.1	Desde GitHub	12
3.2.2	Desde IntelliJ IDEA	14
3.2.3	Comparación ventajas y desventajas	16
3.3	Fetch - Obtener actualizaciones	16
3.4	Merge - Fusionar Ramas	17
3.4.1	Funcionamiento	17
3.4.2	Realizar merge	17
4	Colaboración en proyectos	19
4.1	Pull Requests - Solicitar fusión y merge de cambios	19
4.2	Fork - Crear una copia de un repositorio	22
4.2.1	Creación de un Fork	22
5	Conclusión	24
6	Bibliografía y Recomendaciones	25

1 Introducción

El presente reporte corresponde a la continuación del informe «*[RT-POO-01]GitHub-IntelliJIDEA*» y está diseñado para profundizar en el uso de *IntelliJ IDEA* en la gestión de proyectos con *GitHub*.

Se abordarán las operaciones esenciales de *GitHub* y cómo manejarlas desde *IntelliJ IDEA*, enfocado en el manejo de *Commits*, *Push*, *Pull*, *trabajo con ramas*, *Merge* y *Fork*, agregando una explicación de cada proceso.

Al igual que el primer reporte, el enfoque estará en el uso de *IntelliJ IDEA*, sin detallar el funcionamiento de *Git*, pero proporcionando la información necesaria para comprender y aplicar cada operación de manera efectiva.

Todo el contenido técnico y las definiciones relacionadas con *Git* en este informe están basados en el libro «*Pro Git*».

Finalmente, se asumirá que los pasos del primer reporte fueron ejecutados y que el ambiente está preparado para la interacción de *IntelliJ IDEA* con el repositorio vinculado, utilizando el código generado en el reporte *[RT-POO-01]*.

2 Manejo de los cambios en un repositorio

2.1. Commit - Registrar cambios locales

Un «**commit**» en *Git*, guarda un registro del estado actual de un proyecto en el repositorio local. Este registro es comparable a una fotografía del proyecto en un momento específico, lo que permite conservar una copia exacta de los archivos en ese instante.

Cada *commit* genera un punto en el historial del proyecto, lo que facilita el seguimiento de los cambios realizados y permite regresar a versiones anteriores si es necesario.

Nota

Los cambios en el proyecto no se incorporan al historial de versiones del repositorio hasta que se lleva a cabo un *commit*, momento en el cual quedan registrados.

2.1.1 «Commit» según Pro Git

«La confirmación guarda una instantánea de tu área de preparación...Cada vez que realizas un commit, guardas una instantánea de tu proyecto la cual puedes usar para comparar o volver a ella luego.»

2.1.2 Realizar Commit

Para ver el proceso de un *commit*, se añadió un método que determina si un número es primo al código generado en [RT-POO-01].

```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println("Manejando el repositorio desde GitHub");
5
6         // ---- esPrimo ----
7         int num = 5;
8         String resultado = esPrimo(num)
9             ? "El número " + num + " es primo"
10            : "El número " + num + " NO es primo";
11         System.out.println(resultado);
12     }
13
14     public static boolean esPrimo(int num) {
15         for (int i = 2; i < num; i++) {
16             if (num % i == 0){
17                 return false;
18             }
19         }
20         return true;
21     }
22 }
```

Figura 1: Código nuevo

1. En la sección de «Commit», ubicada en la columna derecha se seleccionan los archivos modificados, eliminados o agregados a incluir.

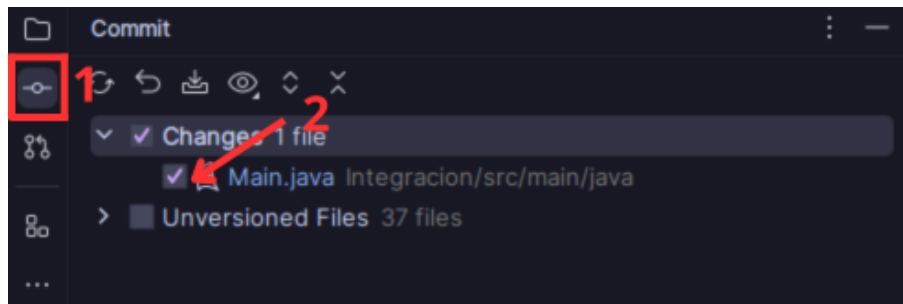


Figura 2: Sección «Commit»

2. Se debe agregar un *mensaje de commit* obligatorio. Finalmente, para completar el proceso se presiona el botón *Commit*.

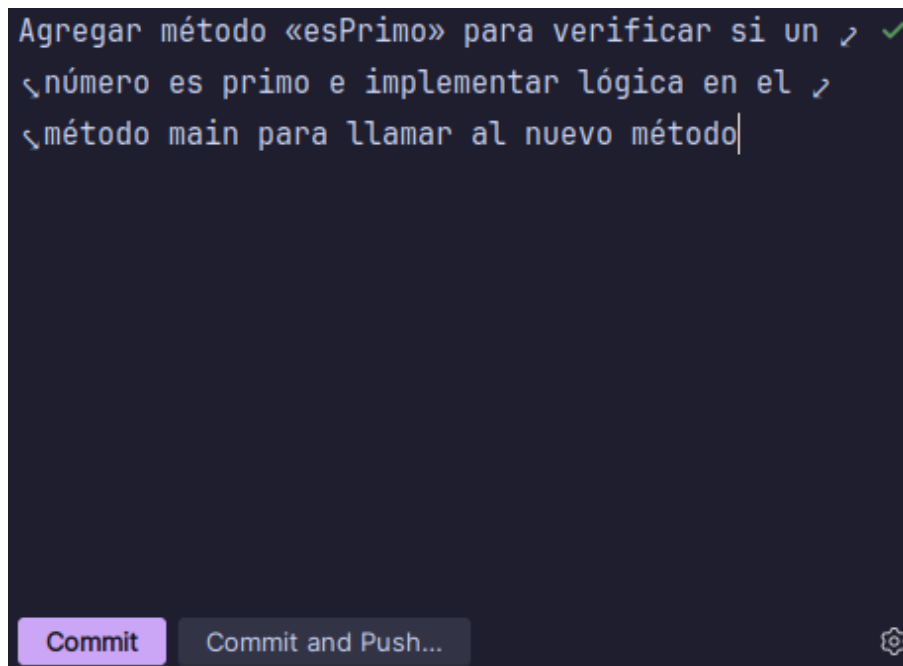


Figura 3: Mensaje de «Commit»

Nota

Por defecto, **IntelliJ IDEA** ofrece dos opciones al realizar un commit:

- **Commit:** Registra los cambios seleccionados en el repositorio local, guardándolos en el historial de versiones local, pero sin enviarlos al remoto (nube).
- **Commit y Push:** Registra los cambios en el repositorio local y los envía al remoto (nube) en un solo paso, simplificando la sincronización con el central.

Por ahora, se realizará solo el *Commit* para mantener los cambios localmente.

2.1.3 Revisar cambios - Diff

En la sección *Commit*, al hacer clic derecho sobre un archivo y seleccionar «*Show Diff*», se visualizarán las diferencias entre el commit anterior y el estado actual del proyecto. Esto permite revisar los cambios antes de confirmarlos, asegurando un mejor control de versiones.

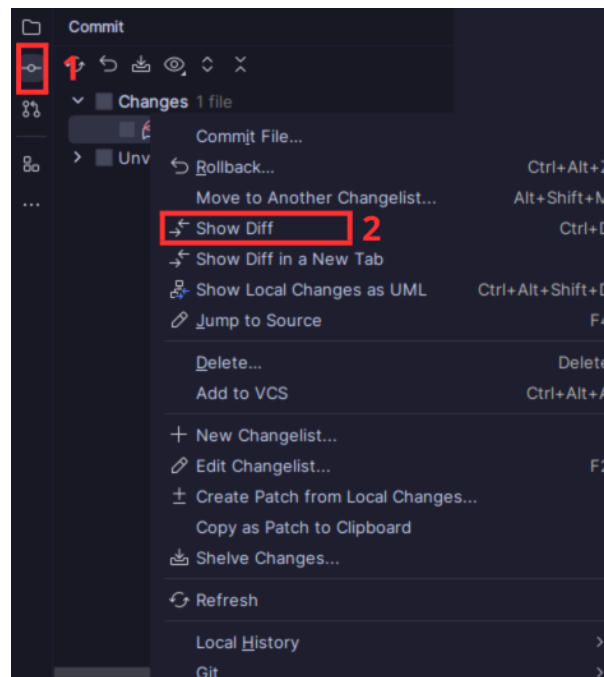


Figura 4: «*Show Diff*» en sección «*Commit*»

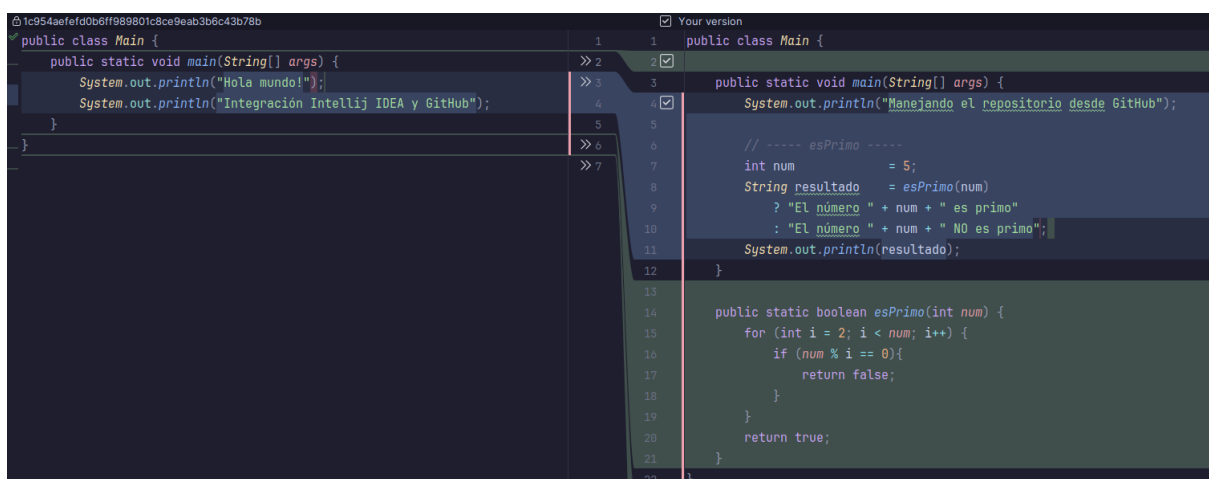


Figura 5: Interfaz «*Diff*»

2.1.4 Commit Message

El mensaje de un «*Commit*» es obligatorio y debe describir de manera concisa los cambios realizados, siendo su propósito facilitar la comprensión del alcance y la intención de las modificaciones, manteniendo un historial claro y organizado del proyecto.

Convención utilizada

*Por convención, los mensajes de commit se escriben en **tercera persona** y en **infinitivo**, facilitando la coherencia y legibilidad en el historial de versiones.*

Nota

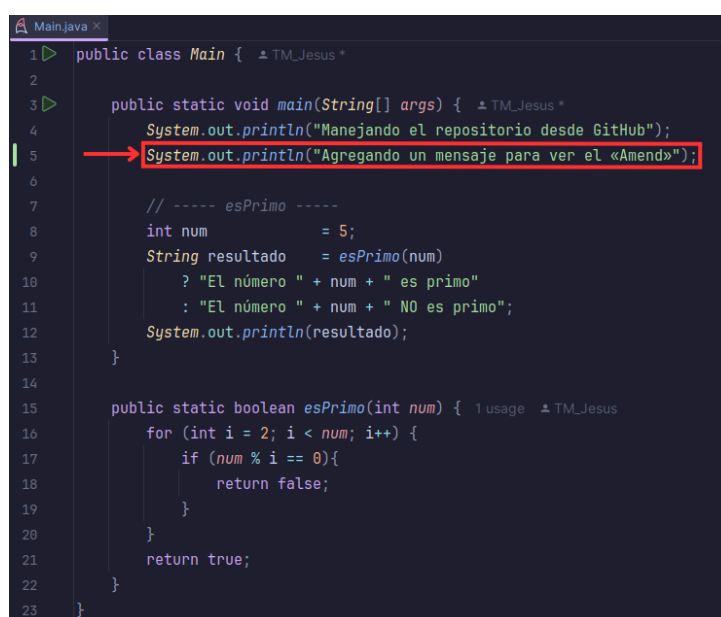
Adicionalmente, existe la convención de «*Conventional Commits*», que proporciona una estructura clara para redactar mensajes de «*Commit*» en proyectos de software. Por ejemplo, siguiendo esta convención el mensaje de commit en la *Figura 3* sería:

- *feat: agregar método esPrimo para verificar números primos*

Implementar llamado el método main al nuevo método esPrimo.

2.1.5 Amend - Modificar el último Commit

Si es necesario modificar el último *Commit*, ya sea para agregar cambios, corregir el mensaje o solucionar un error, se debe utilizar la opción «*Amend*». Esta opción permite reemplazar el último *Commit* con una versión actualizada sin generar uno nuevo.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println("Manejando el repositorio desde GitHub");
5         System.out.println("Agregando un mensaje para ver el «Amend»");
6
7         // ---- esPrimo ----
8         int num = 5;
9         String resultado = esPrimo(num);
10        ? "El número " + num + " es primo"
11        : "El número " + num + " NO es primo";
12        System.out.println(resultado);
13    }
14
15    public static boolean esPrimo(int num) {
16        for (int i = 2; i < num; i++) {
17            if (num % i == 0) {
18                return false;
19            }
20        }
21        return true;
22    }
23 }
```

Figura 6: Nueva línea de código

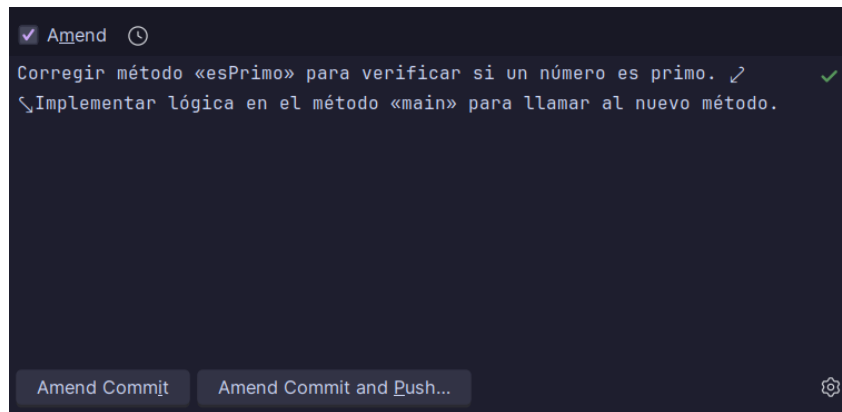


Figura 7: Aplicando Amend

2.2. Push - Subir cambios al remoto

Realizado los cambios y registrados en el repositorio local mediante un *Commit*, es necesario subir estos cambios al repositorio remoto en *GitHub* para que estén disponibles para otros colaboradores. Este proceso se realiza mediante un «**Push**».

El «**Push**» envía los commits del repositorio local (máquina local) al remoto (nube), sincronizando ambos y asegurando que los cambios estén disponibles en línea.

2.2.1 «Push» según Pro Git

«Cuando tienes un proyecto que quieres compartir, debes enviarlo a un servidor. El comando para hacerlo es simple: `git push`»

2.2.2 Realizar Push

- En la barra de herramientas se selecciona Git > Push.

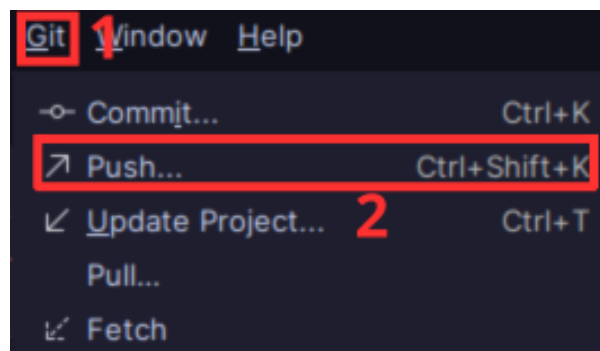


Figura 8: Despliegue de opciones «Git»

- Dentro de la opción «*Push*», aparecerá una interfaz que permite revisar los cambios

que se enviarán al repositorio remoto, las ramas involucradas y el destino del *Push*.

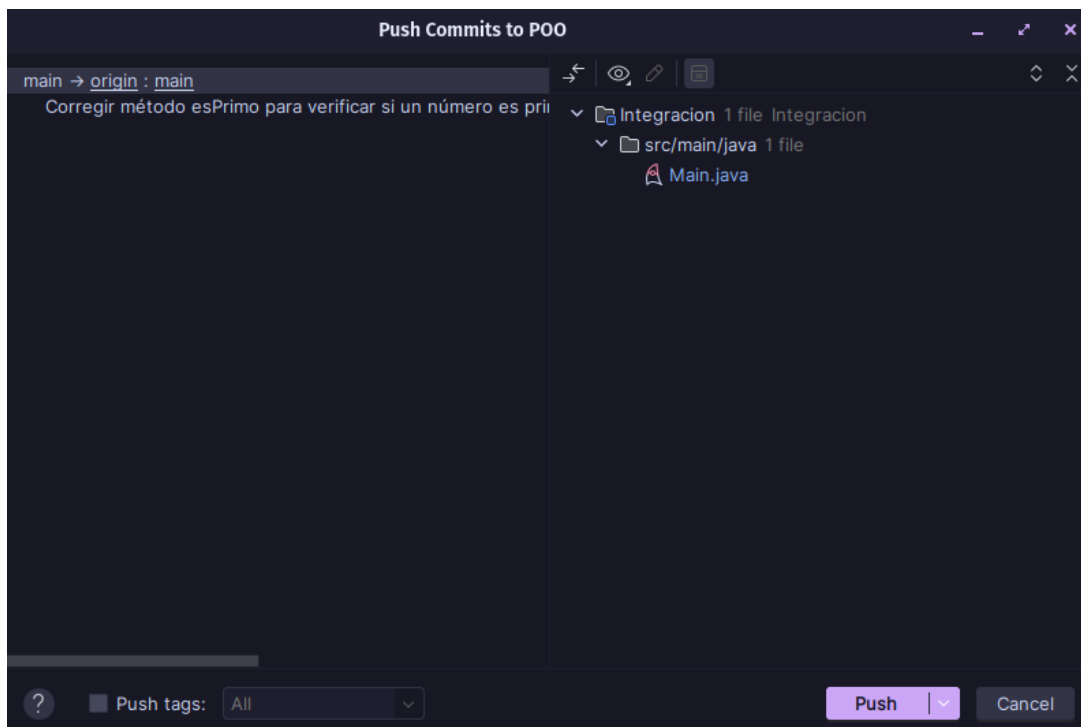


Figura 9: Interfaz de «Push»

- Finalmente, si se compara el estado del repositorio antes del «Push», este permanecerá sin cambios. En contraste, tras aplicar el «Push» el commit anterior aparecerá en el repositorio remoto.

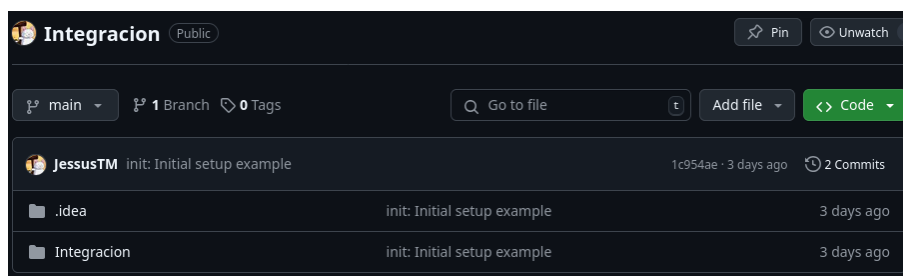


Figura 10: Estado inicial repositorio remoto

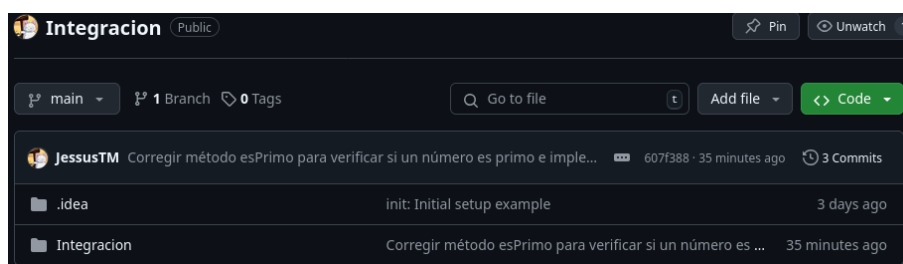


Figura 11: Estado del repositorio remoto después del «Push»

2.3. Pull - Traer cambios del remoto

Cuando se trabaja en un proyecto con múltiples colaboradores, el repositorio remoto en *GitHub* puede experimentar varios cambios a lo largo de su evolución. Para asegurarse de que se está utilizando la versión más reciente del proyecto, es necesario sincronizar el repositorio remoto con el local mediante un **«Pull»**.

El *«Pull»* trae los cambios más recientes del repositorio remoto e integra esos cambios en el repositorio local.

2.3.1 «Pull» según Pro Git

«Git pull, automáticamente combinará la rama maestra remota con tu rama local, luego de haber traído toda la información de ella. También lista todas las referencias remotas de las que ha traído datos»

2.3.1. Realizar «Pull»

- Para realizar un *Pull* en la barra de herramientas se selecciona Git > Pull en el menú desplegable.

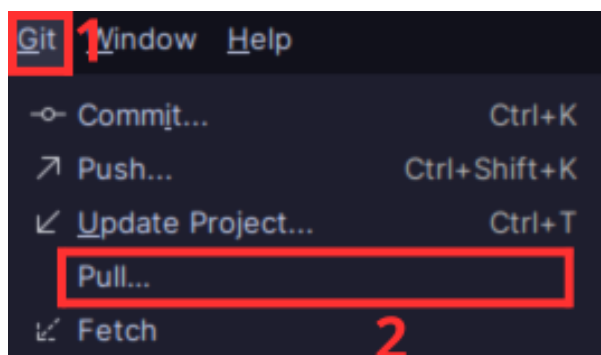


Figura 12: Despliegue de opciones *Git*

- En la ventana *«Pull to rama»*, se elige el origen remoto (origin) y la rama (main) de la cual se quiere descargar los cambios. Alternativamente, el usuario tiene la opción de modificar más parámetros mediante *«Modify options»* antes de confirmar la operación presionando el botón *«Pull»*.

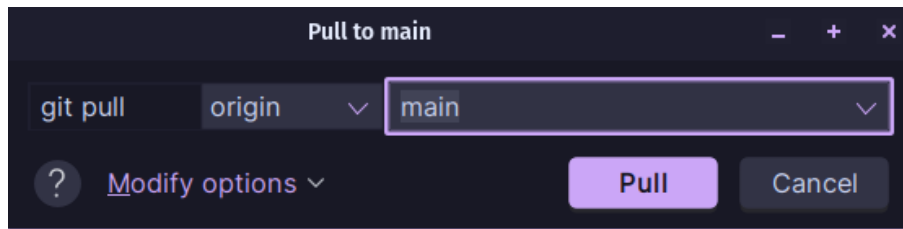


Figura 13: Pull to *[rama]*

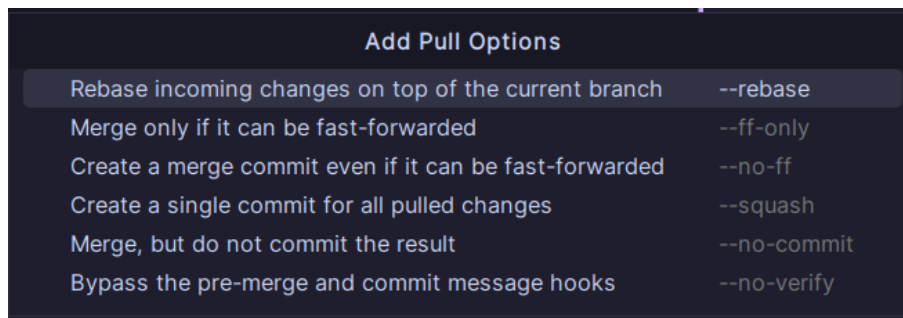


Figura 14: Modify options

- Al hacer click en el botón «Pull», se descargarán todos los cambios y registros del repositorio remoto. Para confirmar que la operación fue exitosa, aparecerá un mensaje en la esquina inferior derecha de la pantalla.

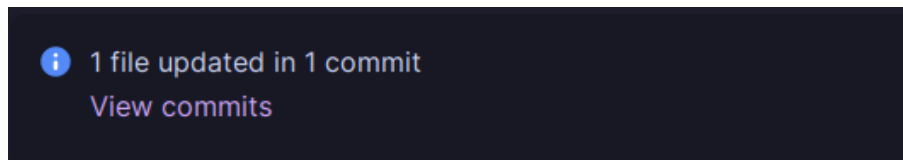


Figura 15: Confirmación del «Pull»

- En la sección «View commits», se pueden visualizar los detalles de los commits descargados, incluyendo su historial y los cambios realizados en cada uno.

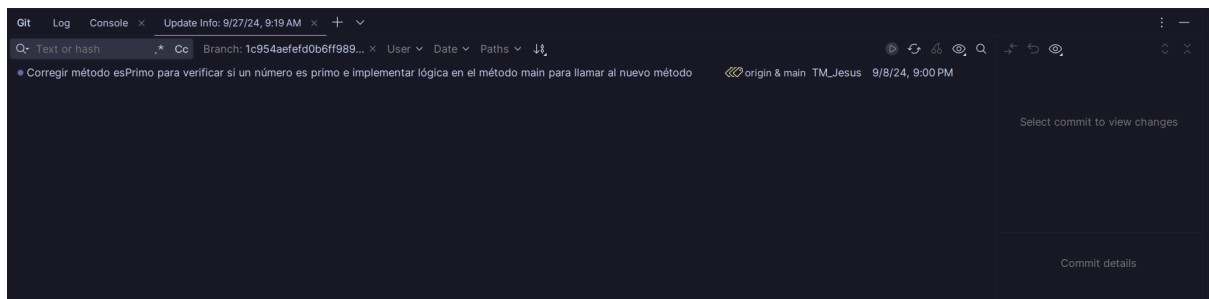


Figura 16: Visualización de los «Commits» recientes

3 Trabajo con ramas

3.1. ¿Qué es una rama?

Una rama en *Git* es una copia independiente del proyecto en un estado específico, derivada de otra rama, generalmente la principal. Facilita el desarrollo de cambios sin alterar la versión estable. Por defecto, la primera rama se denomina `master`, aunque su nombre puede modificarse según las preferencias del equipo.

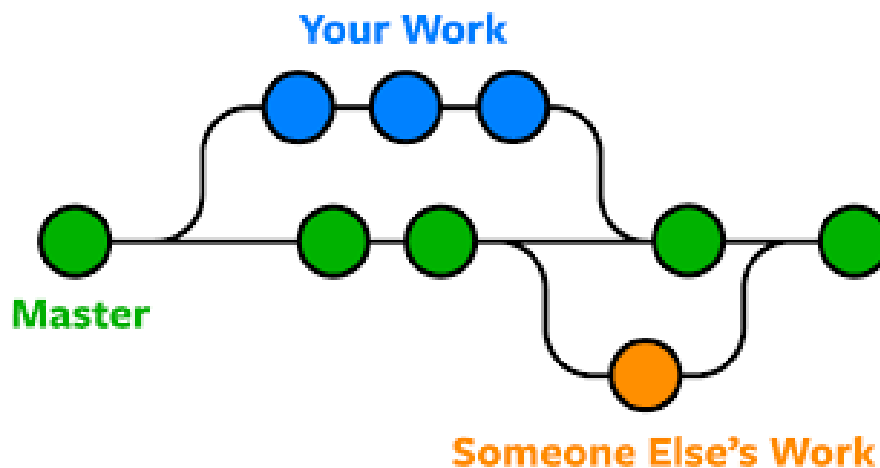


Figura 17: Ramas en Git

Nota

La rama «*master*» en *Git* no es una rama especial, sino una rama como cualquier otra. La única razón por la cual aparece en la mayoría de los repositorios es porque GitHub la crea por defecto al crear el repositorio.

3.2. Crear rama

Para crear una rama en un repositorio, este proceso se puede realizar desde *IntelliJ IDEA* o directamente desde *GitHub*. Sin embargo, es preferible hacerlo desde este último al dar una gestión centralizada y visibilidad inmediata de las ramas y su historial.

3.2.1. Desde GitHub

- Para crear una rama desde *GitHub*, dentro de la página principal del repositorio se selecciona «**Branch**».

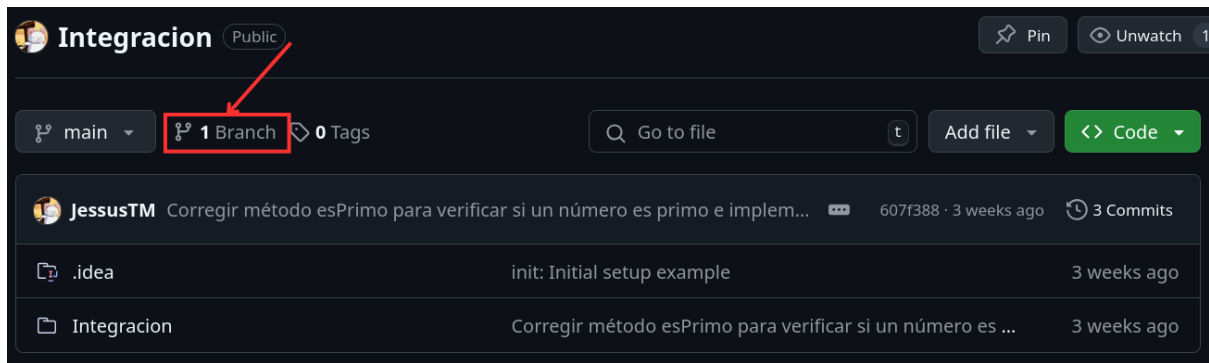


Figura 18: Sección «Branch» de GitHub

- Dentro de la interfaz se mostrarán todas las ramas del repositorio. Para crear la rama se deberá seleccionar «**New branch**». Adicionalmente, al hacer clic en los tres puntos junto a cada rama, es posible renombrarla o visualizar su actividad, incluyendo el registro de *Commits*.

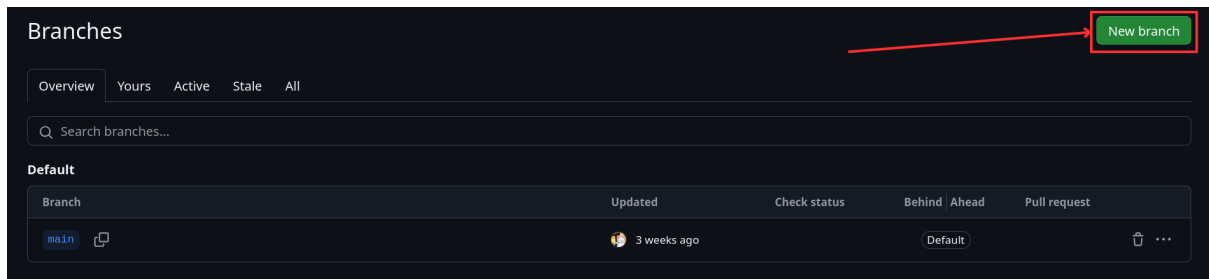


Figura 19: Branches

- En la ventana «**Create a branch**», se puede definir el nombre de la nueva rama y seleccionar la rama base de la cual se creará.

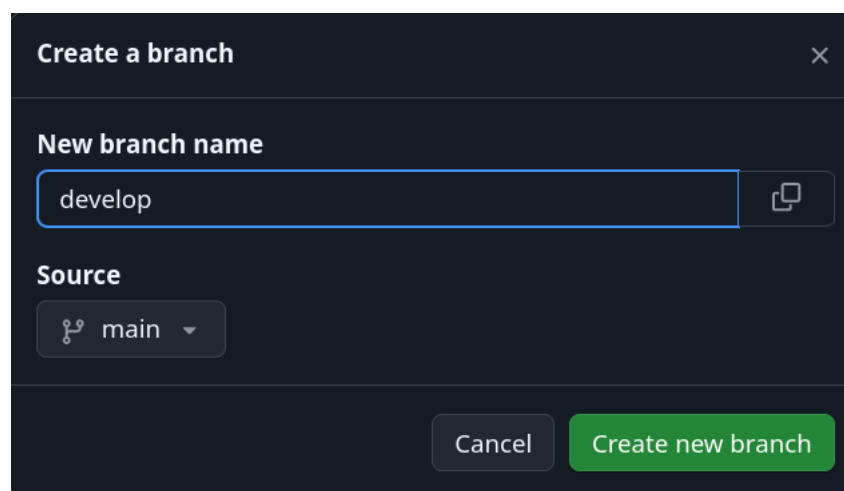


Figura 20: Ventana para la creación de una nueva rama

- Una vez creada la rama, en la sección «**Branches**» aparecerán todas las ramas creadas, además de la rama por defecto del repositorio.

Default				
Branch	Updated	Check status	Behind / Ahead	Pull request
main	3 weeks ago		Default	...
Your branches				
Branch	Updated	Check status	Behind / Ahead	Pull request
develop	now		0 0	...

Figura 21: Lista de ramas creadas en el repositorio

3.2.2. Desde IntelliJ IDEA

- Para crear una rama en *IntelliJ IDEA*, se debe seleccionar el menú desplegable junto al nombre del proyecto o ir a la barra de herramientas y elegir *Git > Branches*. Luego, se debe hacer clic en la opción «**New Branch**».

Nota

La nueva rama se generará a partir de la rama activa en el momento de su creación, tomando esta como base.

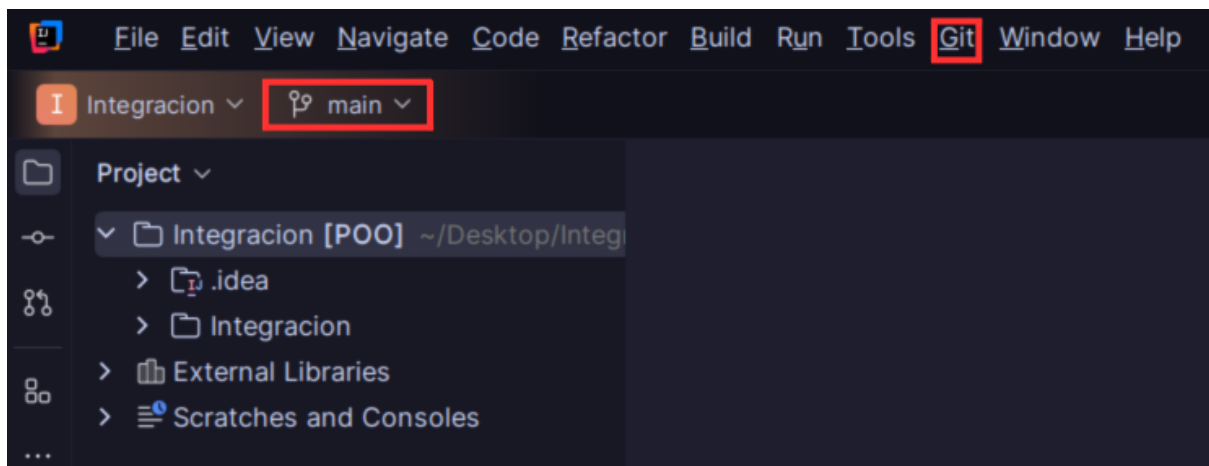


Figura 22: Opciones

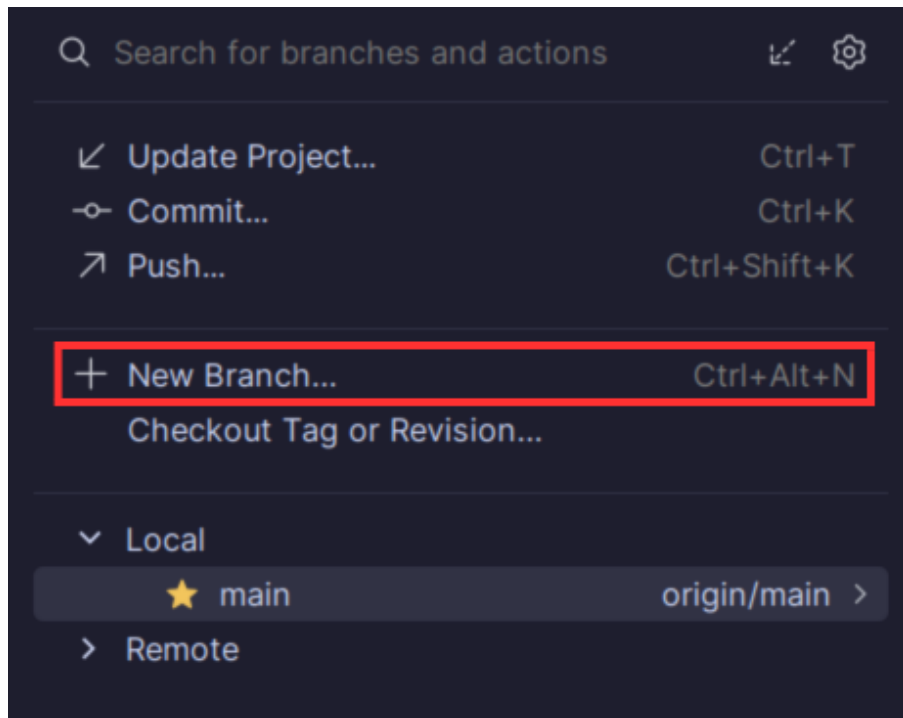


Figura 23: Opción para crear una nueva rama

- En la ventana «**Create New Branch**», se debe ingresar el nombre de la nueva rama. Además, se pueden marcar las opciones «**Checkout branch**» para cambiar a la nueva rama inmediatamente después de crearla, o «**Overwrite existing branch**» para reemplazar una rama existente con el mismo nombre. Finalmente, se debe seleccionar «**Create**» para generar la nueva rama.

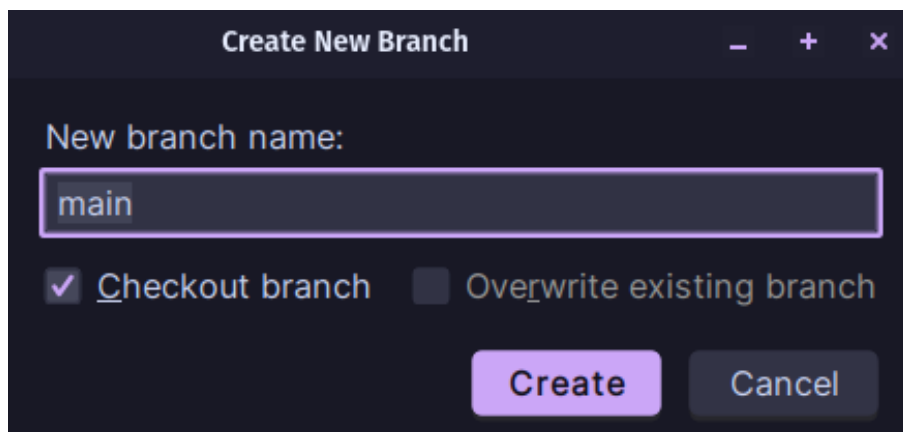


Figura 24: Ventana para la creación de una nueva rama con opciones adicionales

3.2.3. Comparación ventajas y desventajas

Aspecto	Desde IntelliJ IDEA	Desde GitHub
Ventajas	<p>Integración directa con el flujo de trabajo de desarrollo.</p> <p>Permite hacer «checkout» de la rama inmediatamente después de crearla.</p> <p>Acceso rápido desde la interfaz del <i>IDE</i> sin necesidad de cambiar de herramienta.</p> <p>Opción para sobrescribir una rama existente.</p>	<p>Interfaz visual intuitiva para la gestión de ramas.</p> <p>Permite gestionar y visualizar ramas en remoto de forma clara.</p> <p>Útil para colaboradores que no están trabajando directamente en el código (como revisores de código).</p> <p>Se puede crear la rama sin necesidad de tener el código en local.</p>
Desventajas	<p>Requiere tener el proyecto configurado en «<i>IntelliJ IDEA</i>» para poder acceder a las ramas.</p> <p>Menor visibilidad de las ramas remotas si no se hace un «<i>fetch</i>» o «<i>pull</i>» regularmente.</p>	<p>No permite trabajar directamente en la nueva rama sin clonar o hacer «<i>pull</i>» en local.</p> <p>No está integrado directamente en el flujo de trabajo del código en local.</p>

Cuadro 1: Comparación entre la creación de ramas en «IntelliJ IDEA» y «GitHub»

3.3. Fetch - Obtener actualizaciones

Con «*Fetch*» se actualiza los metadatos del repositorio remoto en la copia local *sin modificar la rama actual*. No descarga archivos ni fusiona cambios, solo permite revisar actualizaciones antes de aplicarlas.

Nota

En contraste al *fetch*, el *pull* combina *fetch* y *merge*, trayendo los cambios remotos e integrándolos automáticamente en la rama actual.

- **Actualizar referencias remotas:** Actualiza las referencias de las ramas remotas, como origin/main y origin/develop, en el repositorio local.
- **No modifica la rama actual:** A diferencia de «**pull**», «**fetch**» no fusiona los cambios con la rama actual, permitiendo controlar cuándo y cómo se integran.
- **Uso común:** Se usa antes de un merge para revisar los cambios remotos antes de

decidir si integrarlos.

3.4. Merge - Fusionar Ramas

El «Merge» se utiliza para integrar los cambios realizados en una rama a otra. Se usa cuando una rama ya ha completado la funcionalidad en la que se trabaja y se desea incorporar sus modificaciones a alguna rama, el **merge** permite combinar las líneas de desarrollo.

3.4.1. Funcionamiento

El «Merge» en «Git» toma dos ramas de desarrollo, cada una con su secuencia de «Commits» y sigue los siguientes pasos:

1. **Buscar un commit base común:** Git identifica el commit más reciente que ambas ramas tienen en común, el cual es el punto de partida donde comenzaron a divergir.
2. **Comparar cambios:** Git compara los cambios realizados en cada rama desde el commit base común, permitiendo entender las modificaciones en cada una.
3. **Crear un nuevo commit de merge:** Basado en las diferencias, Git crea un nuevo commit de merge en la rama de destino. Este commit tiene dos padres: el último commit de la rama de destino y el último commit de la rama que se está fusionando.

El resultado es un historial de commits que refleja todas las modificaciones realizadas en ambas ramas, ofreciendo una visión completa y continua del desarrollo del proyecto. Este historial muestra cómo se han integrado los cambios de distintas ramas en un solo flujo.

3.4.2. Realizar merge

- En la ventana «Merge into *rama*», se podrá fusionar los cambios de una **rama de origen hacia la rama actual**. Se debe escoger el nombre de la rama que se desea mezclar, ya que está se integrará en la rama en la que se encuentra trabajando.

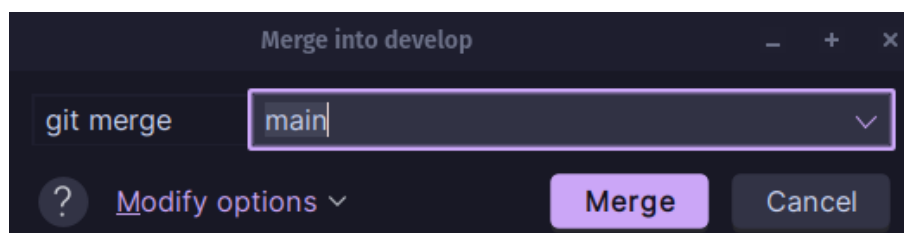


Figura 25: Proceso de selección de ramas para realizar un «Merge»

- Al seleccionar una rama, se desplegarán las ramas locales disponibles en la máquina del usuario, junto con aquellas que tienen el prefijo **origin/**, indicando que son ramas remotas alojadas en «**GitHub**».

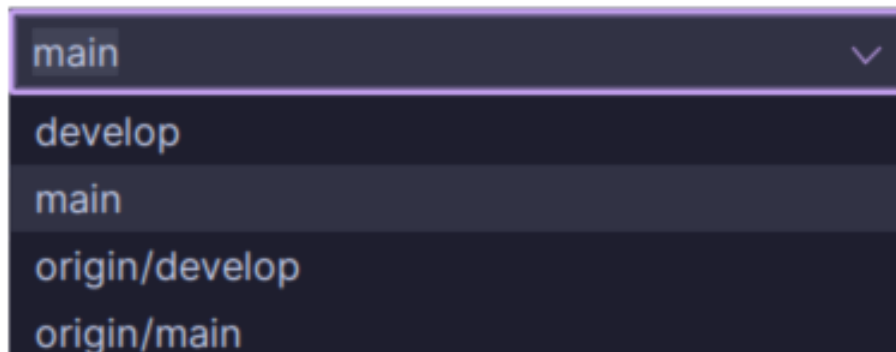


Figura 26: Ramas

- Al hacer click en «**Merge**», aparecerá en la esquina inferior derecha un mensaje de confirmación que indicará cuántos «**Commits**» se agregaron a la rama.

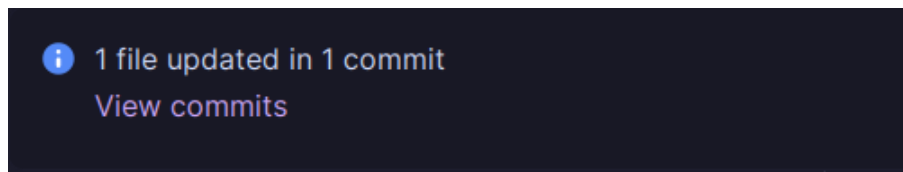


Figura 27: Mensaje de confirmación tras realizar el «**Merge**»

4 Colaboración en proyectos

4.1. Pull Requests - Solicitar fusión y merge de cambios

Un «**Pull Request (PR)**» es una solicitud para fusionar los cambios realizados en una rama a otra. Este proceso permite que los demás colaboradores del repositorio, puedan revisar, comentar y aprobar los cambios antes de que se integren en la rama principal.

- Una vez que se han realizado los cambios en una rama, se debe abrir un «**Pull Request**» para solicitar la integración de esos cambios. En la página del repositorio, aparecerá el botón «**Compare & pull request**» que permitirá crear la solicitud.

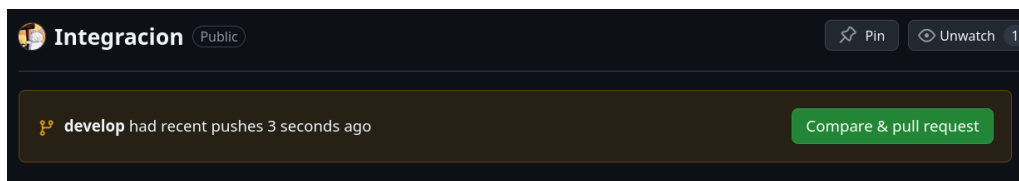


Figura 28: Botón «**Compare & Pull Request**» en GitHub

- Al hacer click en este botón, se redirigirá a una página donde se deberá completar la información del «**Pull Request**». Obligatoriamente se debe especificar un título y descripción que proporcionen una explicación detallada de las modificaciones realizadas, indicando el propósito de los cambios y cualquier aspecto relevante.

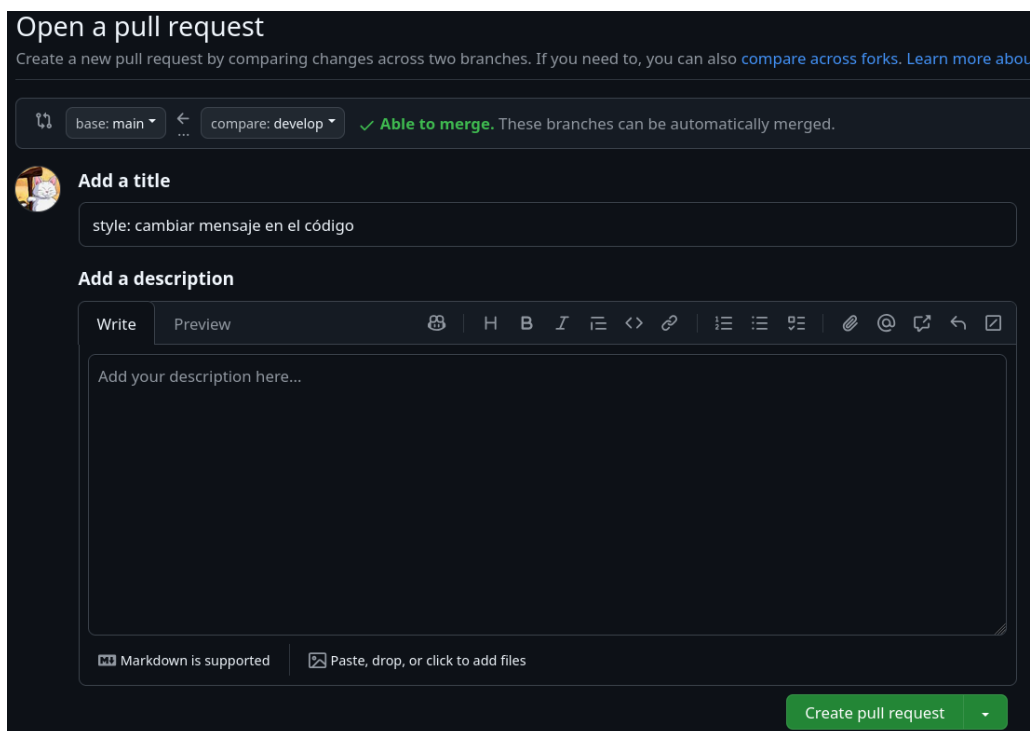


Figura 29: Página de creación del «**Pull Request**»

Nota

Adicionalmente, se puede generar etiquetas que identifiquen el tipo de cambios realizados (como bugfix o feature) y asignar revisores que se encarguen de examinar el código. Una vez que toda la información ha sido completada, se presiona el botón «**Create pull request**» para enviar la solicitud de fusión.

- **GitHub** proporciona una interfaz intuitiva para la revisión, que incluye herramientas como la comparación de diferencias («**diff**») entre la rama principal y la rama del «**Pull Request**». Esto facilita la visualización de los cambios propuestos y permite a los revisores verificar su impacto en el código base.

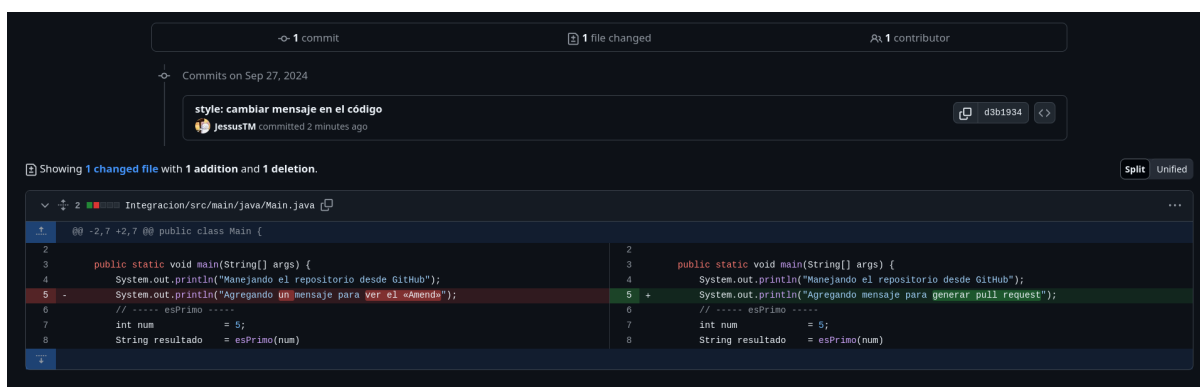


Figura 30: Revisión de un «Pull Request» en GitHub

- Después de que se ha creado el «**Pull Request**», los colaboradores del proyecto pueden revisarlo, proceso el cual incluye la posibilidad de dejar comentarios en el código, sugerir cambios o realizar preguntas para aclarar aspectos específicos. Los revisores pueden aprobar el «**Pull Request**» o solicitar modificaciones antes de que se proceda a fusionar los cambios.

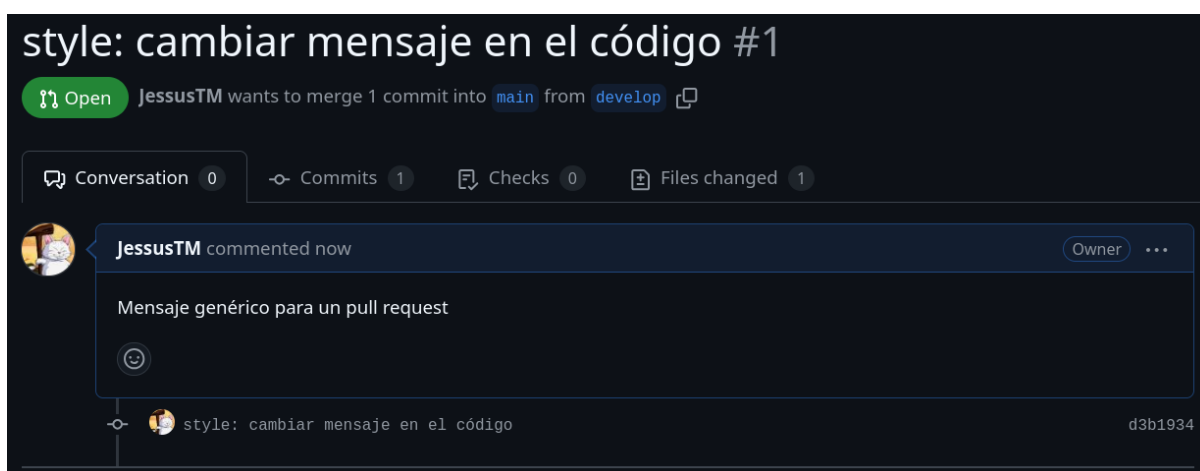


Figura 31: Creación del «Pull Request»

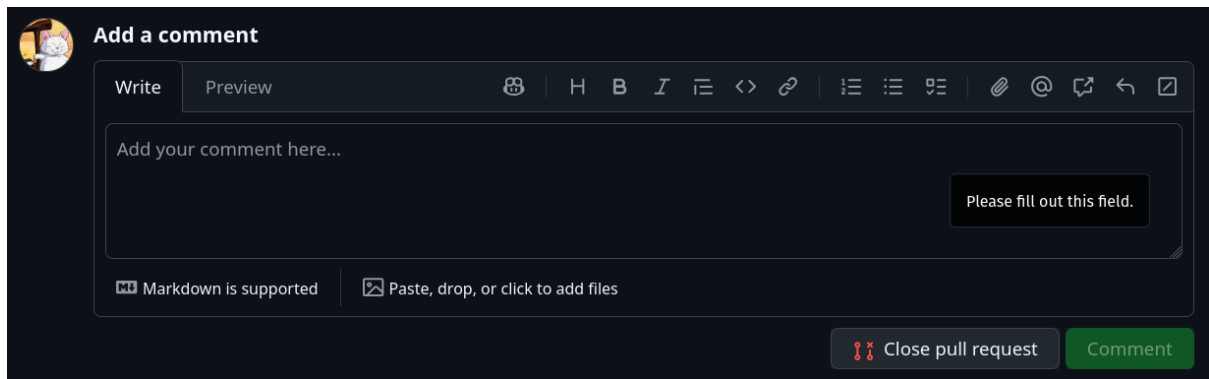


Figura 32: Sección de comentarios del «Pull Request»

- Una vez que los revisores aprueban el «Pull Request», se debe fusionar los cambios en la rama principal del proyecto. Este proceso se realiza presionando el botón «Merge pull request», parece una ventana donde se solicita un mensaje de confirmación. Se debe proporcionar un título que describa la fusión y opcionalmente un cuerpo que detalle cualquier aspecto adicional relevante.

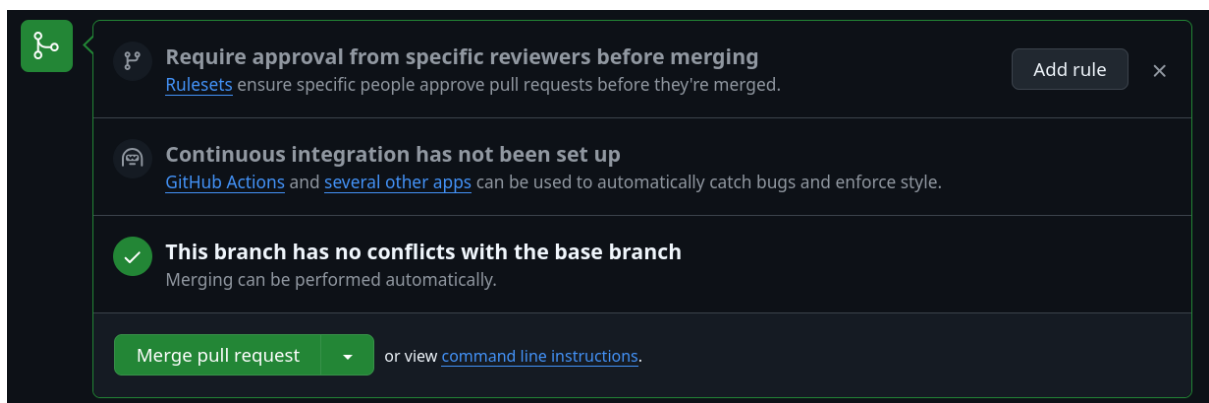


Figura 33: Fusión de un «Pull Request»

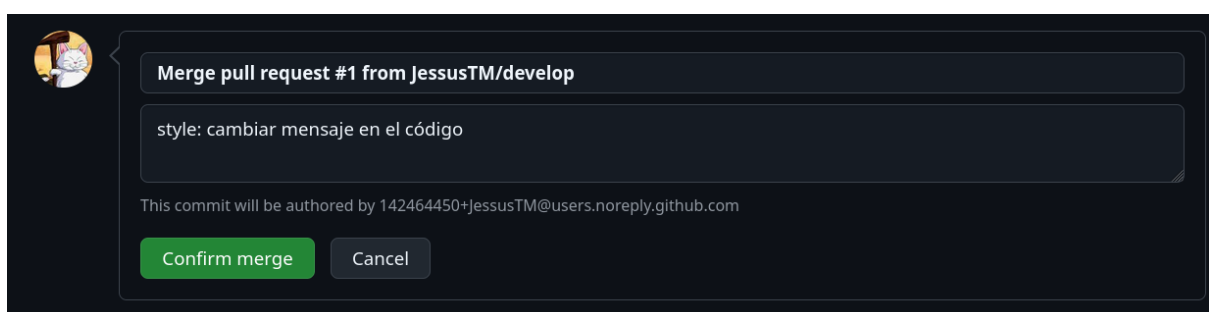


Figura 34: Confirmar «Merge»

- Finalmente, la pantalla actualiza su estado para indicar que el «Pull Request» ha sido exitoso, lo que significa que el «Merge» fue completado y se ha integrado en la rama correspondiente.



Figura 35: «Pull Request» finalizado

4.2. Fork - Crear una copia de un repositorio

El «**Fork**» crea una copia completa de un repositorio en la cuenta del usuario, permitiendo trabajar en el código de manera independiente a el repositorio original.

4.2.1. Creación de un Fork

Para realizar un «**Fork**» de un repositorio, se debe navegar hasta la página principal del repositorio en «**GitHub**» que se desea copiar. En la parte superior derecha de la página, se encuentra el botón «**Fork**». Al hacer click en este botón, «**GitHub**» iniciará el proceso de creación de una copia completa del repositorio dentro de la cuenta del usuario.

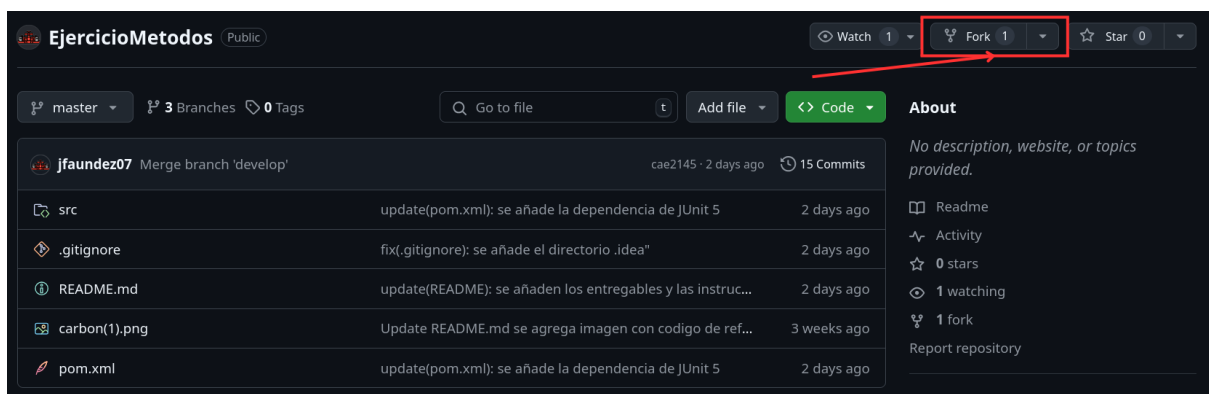
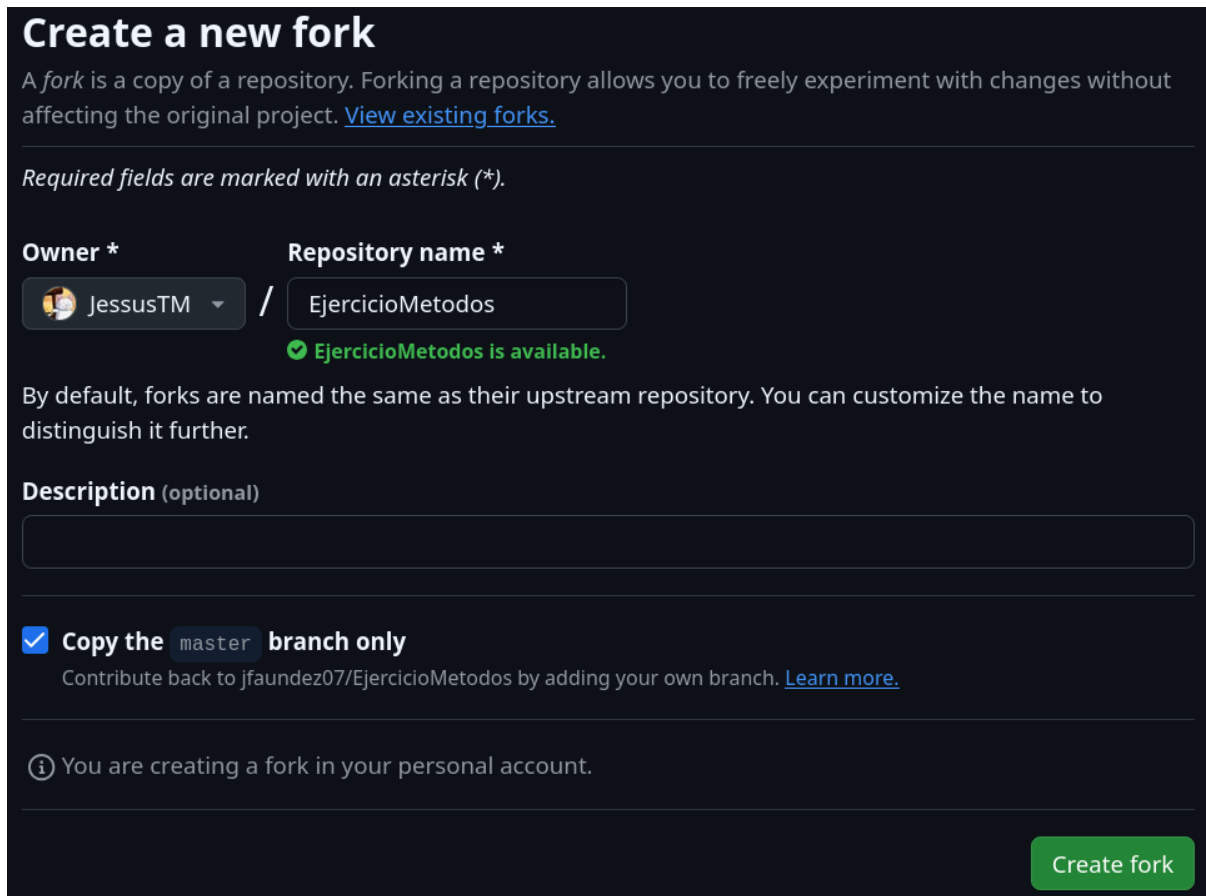


Figura 36: Botón «Fork» en la página principal del repositorio


Al crear un «**Fork**», se debe seleccionar la cuenta de destino donde se almacenará el repositorio. El repositorio quedará disponible en el perfil del usuario que se especifique.



Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (*).


Owner *  JesusTM / **Repository name *** EjercicioMetodos

✓ EjercicioMetodos is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

☒ **Copy the master branch only**
Contribute back to jfaundez07/EjercicioMetodos by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

[Create fork](#)

Figura 37: Proceso de creación de un «**Fork**» en GitHub

Una vez copiado el repositorio, se puede trabajar en la versión independiente aplicando todas las opciones de «**Git**» como clonar el repositorio localmente, realizar cambios y luego enviar esos cambios al repositorio en «**GitHub**».

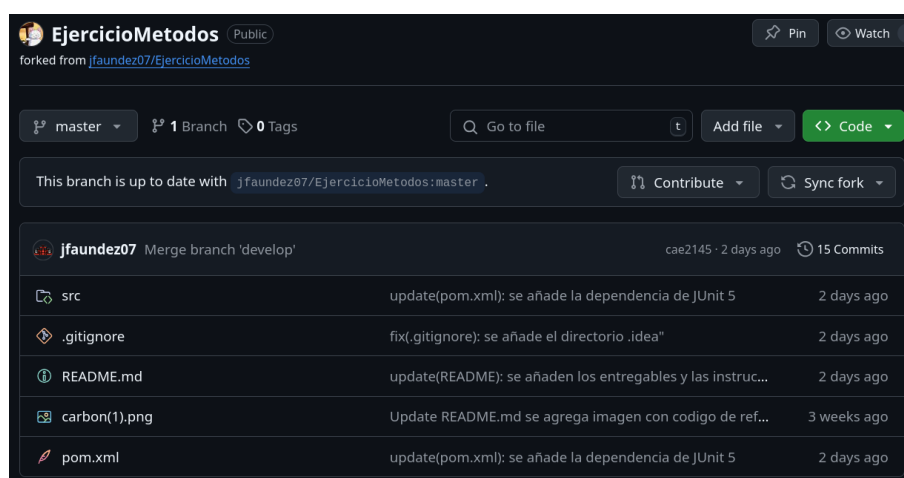


Figura 38: Repositorio clonado

5 Conclusión

A lo largo de este reporte se abordó el proceso de manejar repositorios en «**GitHub**», utilizando «**IntelliJ IDEA**» como **IDE**.

En la sección de «**Manejo de los cambios en un repositorio**», se explicó cómo guardar los cambios de un proyecto mediante un «**Commit**», en un repositorio local alojado en la máquina del usuario, además de la importancia de seguir una convención para la escritura de mensajes de «**Commit**». También se incluyó «**diff**» para visualizar los cambios entre versiones del proyecto, así como la opción de modificar el último commit mediante la función «**Amend**», permitiendo la corrección de errores o actualizar el «**Commit**» sin crear uno nuevo. Con «**Push**», se abordó cómo enviar los cambios locales al repositorio remoto en «**GitHub**» asegurando que las modificaciones queden disponibles para los demás colaboradores. Adicionalmente, se explicó cómo utilizar «**Pull**» para traer los cambios de un repositorio remoto, para que el proyecto local esté actualizado con las contribuciones de otros desarrolladores.

En la sección de «**Trabajo con ramas**», se profundizó en la creación de ramas tanto desde «**GitHub**» como desde «**IntelliJ IDEA**», comparando las ventajas y desventajas de cada enfoque. También se explicó el uso del comando «**Fetch**» para obtener actualizaciones del repositorio remoto sin fusionarlas automáticamente. Además, mediante un «**Merge**» se mostró cómo integrar los cambios de una rama en otra.

En la sección de «**Colaboración en proyectos**», se abordó la funcionalidad de los «**Pull Requests**», como medio para la integración y revisión de cambios en proyectos colaborativos. También se exploró la creación y uso de «**Forks**», permitiendo trabajar de manera independiente en una copia de un repositorio sin afectar el proyecto original.

Este reporte se centró en la comprensión del uso de «**GitHub**» para la gestión de versiones y la colaboración en proyectos de software. Además, con la combinación de conceptos teóricos y su aplicación práctica en un entorno de desarrollo real, se logró la adquisición de habilidades para el manejo de proyectos en equipo. Finalmente, dado que «**Git**» y «**GitHub**» son herramientas fundamentales en el desarrollo de software, se recomienda y queda abierta la invitación a seguir profundizando en su estudio y práctica.

6 Bibliografía y Recomendaciones

- [1] S. Chacon and B. Straub, Pro Git, 2nd ed. Apress, 2014. [Online]. Available: <https://git-scm.com/book/en/v2>
- [2] GitHub, “How to add code to your repository | GitHub beginner tutorial.” [Online]. Available: <https://www.youtube.com/watch?v=g2XjJhrGGg4&list=PL01o9M0BetEFcp4SCWinBdpml9B2U25-f&index=5>
- [3] GitHub, “How to create a pull request in 4 min | GitHub for Beginners 2024.” [Online]. Available: <https://www.youtube.com/watch?v=nCKdihvneS0&list=PL01o9M0BetEFcp4SCWinBdpml9B2U25-f&index=6>
- [4] GitHub, “How to merge a pull request | Introduction to GitHub.” [Online]. Available: <https://www.youtube.com/watch?v=FDXSgyDGmho&list=PL01o9M0BetEFcp4SCWinBdpml9B2U25-f&index=7>