Introducción

¡Hola! 👋 Te damos la bienvenida a un nuevo encuentro.

En las clases anteriores, hemos explorado los fundamentos de Git, cómo realizar cambios en un repositorio local y cómo conectar Git con GitHub. También hemos aprendido a colaborar en proyectos con nuestros compañeros.

Hoy, nos centraremos en la colaboración en Git y GitHub, con un enfoque especial en los "pull requests", las mejores prácticas y cómo trabajar con Visual Studio Code y GitHub.

Aprenderás las herramientas esenciales para colaborar en proyectos de código abierto o con un equipo de desarrolladores.

¡Comencemos!🚀

Pull requests

Continuando con nuestro tema de colaboración en Git y GitHub, nos adentraremos en un aspecto fundamental: los **Pull Requests**. Estas solicitudes **nos permiten incorporar cambios a un repositorio original después de pasar por un proceso de revisión**.

En un entorno laboral, los Pull Requests juegan un papel crucial antes del despliegue de cambios. Este proceso de revisión nos ayuda a prevenir posibles fallos y asegurarnos de que nuestras modificaciones sean seguras y funcionales para todos los usuarios.

A continuación, nos centraremos en comprender cómo crear un Pull Request y cómo utilizarlo de manera efectiva.

¡Manos a la obra!

1. Crear un *nuevo repositorio* en GitHub con un **archivo README** y el contenido que desees agregar.

2. Clonar el **repositorio** en tu computadora local para luego poder trabajar con él.

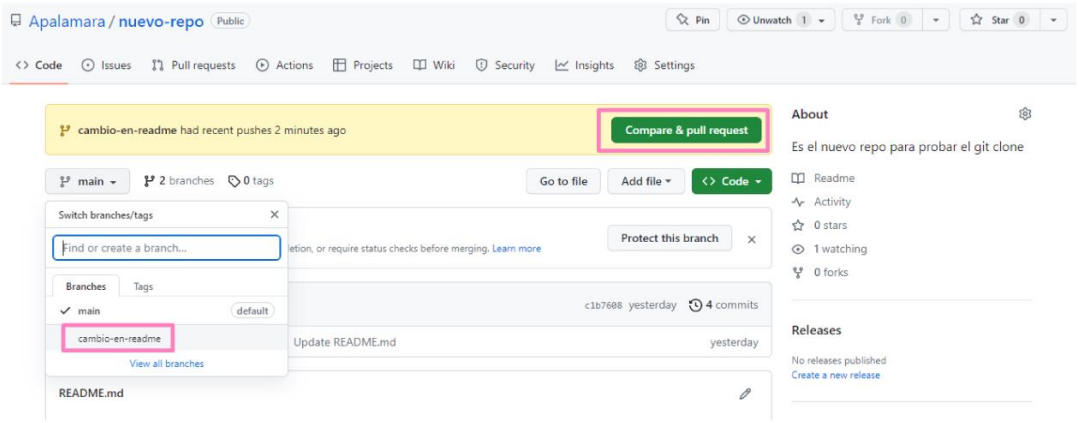
3. Crear una **nueva rama** utilizando Git Bash y cambiar a esa rama.

4. Realizar un cambio en el *archivo README*. Por ejemplo, agregar una línea que diga *"Estoy aprendiendo Pull Requests".*

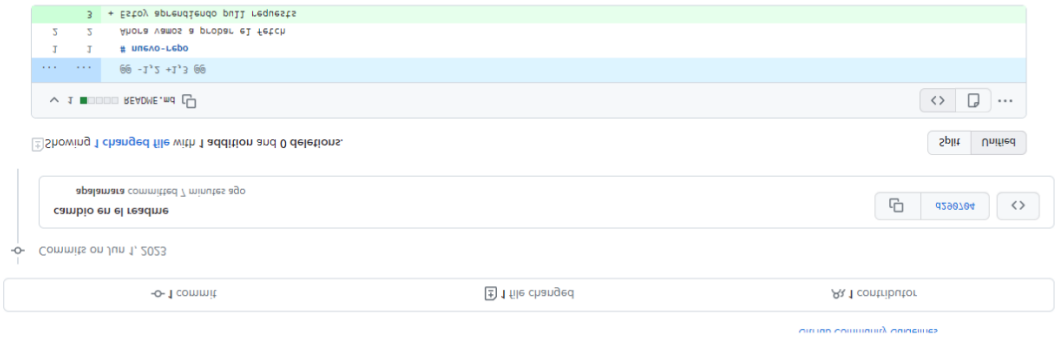
5. Realizar un **commit** para guardar tus cambios.

6. Publicar tu trabajo en GitHub utilizando el comando: **"git push origin [nombre-de-la-rama]"** para enviar los cambios a la rama correspondiente.

En GitHub, notarás que se crea una nueva rama y aparecerá un mensaje que dice *"Compare & pull request"*.



7. Hacer clic en **“Compare & Pull Request”**. Aquí podrás ver la comparación entre la rama que creaste y la rama principal del repositorio. **8.** Tomar un momento para revisar los cambios antes de crear el Pull Request. GitHub te mostrará una vista previa de los cambios realizados, lo cual te permitirá asegurarte de que todo esté correcto antes de enviarlos.



9. Rellenar los detalles del Pull Request, proporcionando un título y una descripción que brinden al propietario del repositorio original una buena idea de tus modificaciones y el motivo detrás de ellas.

10. Finalmente, hacer clic en **“Create Pull Request”.**

¡Listo! 👌 Has tenido la oportunidad de practicar cómo crear un Pull Request, una habilidad esencial para colaborar en proyectos y garantizar la calidad de los cambios antes de su implementación.

En el siguiente vídeo, vamos a poder ver la explicación detallada del paso a paso de la actividad anterior:

💡*Una vez que hayas creado un Pull Request, el propietario del repositorio original podrá revisar tus cambios, realizar comentarios sobre ellos y, eventualmente, fusionarlos en su código. También es posible que te soliciten realizar modificaciones antes de que estén listos para fusionar tus cambios.*

Pull request colaborativo

Ahora, llegó el momento de poner en práctica lo aprendido. El propósito de esta actividad es trabajar con un/a compañer@ para aprender cómo utilizar eficazmente los Pull Requests como una forma de colaboración en GitHub.

Sigue estos pasos para completar la actividad:

!Antes de comenzar, elige un compañero con quien realizar la actividad y comparte los enlaces de sus respectivos repositorios en GitHub.

1. Clonar el repositorio de tu compañero en tu computadora local.

2. Abrir el **repositorio** clonado en tu editor de código preferido.

3. Crear una **nueva rama** (utilizando la consola Git Bash) y cambiarte a esa rama.

4. Realizar cambios en el código o en la documentación del proyecto. Puedes agregar nuevas funcionalidades, corregir errores o realizar mejoras.

6. Realizar **commits** frecuentes a medida que avanzas en tus cambios.

7. Enviar tus cambios al **repositorio remoto** en GitHub.

8. Ir al **repositorio de tu compañero** en GitHub. Deberías ver que se ha creado una nueva rama con tus cambios.

9. Hacer clic en el botón **"Compare & Pull Request"** para iniciar el proceso de solicitud de incorporación de tus cambios al repositorio original.

10. Revisar tus cambios en la vista previa proporcionada por GitHub y, si todo está correcto, proporcionar un título y una descripción clara y concisa para tu Pull Request.

11. Hacer clic en "**Create Pull Request"** para enviar tu solicitud. Luego, comunícate con tu compañero para que revise tu Pull Request, realice comentarios y pueda fusionar tus cambios en su repositorio.

12. Revisar y comentar el Pull Request de tu compañero en su repositorio. Proporcionar sugerencias o mejoras si es necesario.

Una vez que ambos estén satisfechos con los cambios propuestos, el propietario del repositorio original puede fusionar los Pull Requests.

¡Fin de la actividad! 🙌 Has completado la actividad de colaboración con éxito.

Recuerda que la colaboración a través de Pull Requests es una forma efectiva de trabajar en equipo, revisar y mejorar el código, y garantizar la calidad de los cambios antes de su implementación.

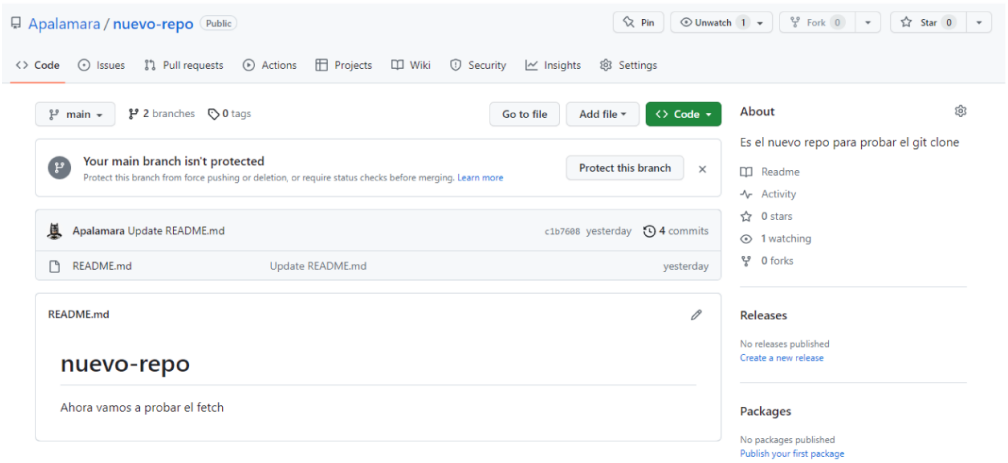
Eliminar repositorios en GitHub

Una vez que hayas creado varios repositorios en GitHub, es posible que en algún momento desees eliminar alguno de ellos. Afortunadamente, este proceso es muy sencillo. Sin embargo, es importante tener en cuenta que **la eliminación de un repositorio es permanente y definitiva**.

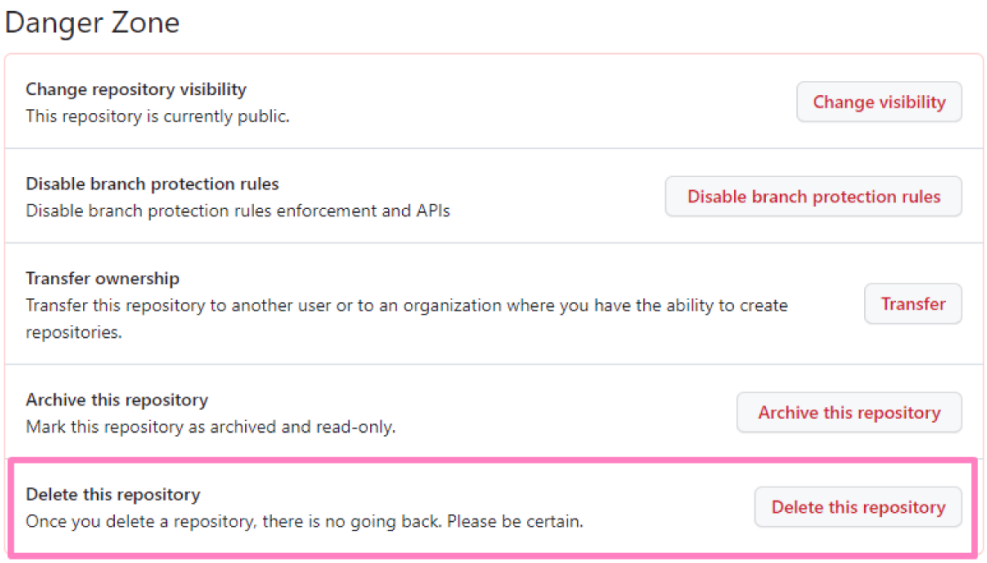
A continuación, sigue el siguiente paso a paso para realizar este procedimiento:

1. Crear un **nuevo repositorio** en GitHub con un *archivo README*.

2. Ir a la página de **"Settings"** (configuración) del repositorio.



3. Desplazarse hacia abajo hasta encontrar la opción de **"Delete this respository"** (eliminar este repositorio).



4. Hacer clic en **“Delete this respository”** y confirmar la acción.

5. Volver a ingresar el nombre del repositorio para confirmar la eliminación.

¡Listo! El repositorio será eliminado de forma definitiva.

Recuerda que al eliminar un repositorio, se eliminarán todos los archivos, historial de cambios y configuraciones asociadas a él. Asegúrate de seleccionar el repositorio correcto antes de proceder, o bien, te recomendamos hacer una copia local antes de eliminar el repositorio en GitHub.

Continuemos explorando y aprendiendo más sobre Git y GitHub en las siguientes secciones. ¡Adelante!

Git ignore

El archivo **.gitignore** es una herramienta poderosa que nos **permite indicar a Git qué archivos o directorios no rastrear ni incluir en nuestro repositorio**.

Esto es útil para evitar el seguimiento de archivos innecesarios o sensibles que no forman parte esencial de nuestro proyecto.

Algunos ejemplos comunes de archivos y directorios que suelen ser ignorados son:

- Archivos muy grandes en tamaño que no son esenciales para el funcionamiento del código.
- Archivos locales temporales que se crean al momento de trabajar, pero que luego se eliminan.
- Información sensible o personal, como contraseñas, claves de acceso, archivos de configuración local, etc.

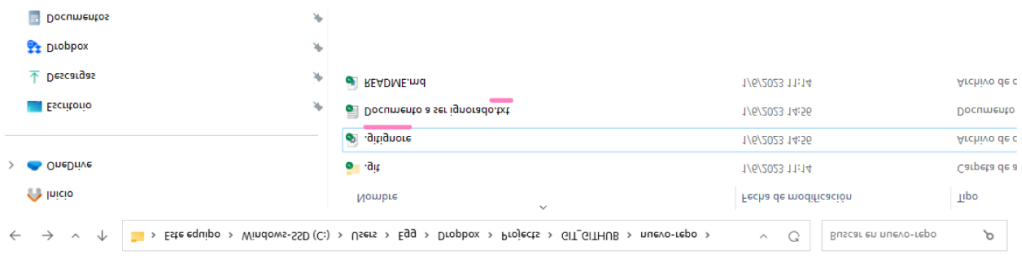
Ahora es el momento de crear nuestro archivo **.gitignore** y especificar qué archivos y directorios deseamos ignorar en nuestro repositorio.

¡Manos a la obra!

1. Elegir un repositorio de los que hemos estado trabajando.

2. Identificar o seleccionar una carpeta o archivo que creamos que deba ser ignorado. Podemos crear, por ejemplo, un nuevo archivo de texto que se llame *"passwords"* y guardar contraseñas falsas para practicar con este archivo.

3. Crear un nuevo archivo de texto (distinto al anterior) llamado **".gitignore"**. No debe tener ninguna otra extensión como .txt.



4. Dentro del archivo *.gitignore*, colocar el nombre del archivo que queremos que sea ignorado. De esta manera:

Ignorar mis archivos

documento_a_ser_ignorado.txt

5. Guardar el archivo .gitignore, publicar los cambios y revisar qué pasa con el archivo que debemos ignorar.

¡Fin de la actividad! A partir de este momento, Git ignorará ese archivo específico y no lo incluirá en el control de versiones ni en las operaciones relacionadas, como git add o git commit.

Recuerda revisar y actualizar regularmente tu archivo .gitignore a medida que tu proyecto evoluciona o se generen nuevos archivos que no deseas rastrear.

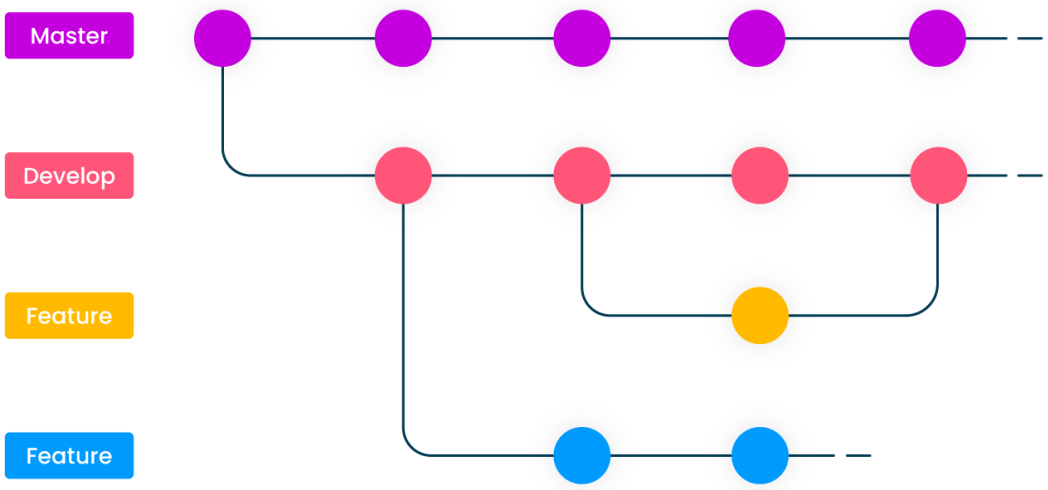
Git Workflow

El **Git Workflow**, o flujo de trabajo de Git, es una forma estandarizada de trabajar con Git que nos permite ser eficientes y productivos en nuestros proyectos.

Este flujo define cómo agregar nuevas funcionalidades al proyecto, probarlas y fusionarlas sin generar conflictos con el trabajo existente.

Para que un flujo de trabajo sea considerado exitoso, debe ofrecer:

- Facilidad para deshacer cambios en caso de errores.
- Escalabilidad, adaptándose al tamaño del equipo.
- Fácil comprensión para que todos los miembros del equipo se adapten rápidamente.



Ahora, imagina que estás liderando un equipo de desarrolladores en un proyecto de software. Para mantener un flujo de trabajo organizado y evitar conflictos, deciden utilizar el flujo de ramas en Git.

¡Pongámoslo en práctica!

1. Crear la rama "develop".

Comienza creando una *rama principal* llamada **"develop**" en tu repositorio. Esta rama será el punto de partida para el desarrollo continuo del proyecto.

2. Crear una nueva rama "feature".

Un desarrollador necesita trabajar en una nueva funcionalidad, como agregar un formulario de registro al proyecto. En lugar de hacer cambios directamente en la rama *"develop"*, el desarrollador crea una *nueva rama* llamada **"feature/registration-form"** a partir de la rama *"develop"*.

3. Trabajar en la nueva rama "feature".

El desarrollador cambia a la rama *"feature/registration-form"* y comienza a trabajar en la implementación del formulario de registro. Agrega los archivos necesarios, escribe el código y realiza los cambios que considera necesarios.

4. Realizar commits en la rama "feature".

A medida que el desarrollador avanza en la implementación del formulario de registro, realiza commits en la rama *"feature/registration-form"* para registrar los cambios realizados.

5. Probar y revisar los cambios en la rama "feature".

Una vez que el desarrollador ha completado la implementación del formulario de registro y ha realizado los *commits* correspondientes, es hora de probar los cambios localmente. Esto asegura que todo funcione correctamente. Si se encuentran problemas, se pueden realizar más modificaciones y commits para reflejar esos cambios.

6. Fusionar la rama "feature" con la rama "develop".

Después de realizar pruebas y revisar los cambios en la rama *"feature/registration-form"*, el desarrollador está listo para incorporar la nueva funcionalidad a la rama *"develop"*. Se realiza una *fusión* (merge) de la rama *"feature/registration-form"* con la rama *"develop"* para integrar los cambios en la rama principal del proyecto.

7. Eliminar la rama "feature".

Una vez completada la fusión y con los cambios de la funcionalidad del formulario de registro en la rama "develop", la rama "feature/registration-form" ya no es necesaria y puede ser eliminada.

¡Fin de la actividad! 🤖 Este fue un ejemplo básico del workflow de ramas en Git utilizando las ramas "develop" y "feature".

Con este enfoque, los desarrolladores trabajaron en ramas de características específicas y fusionaron sus cambios en la rama "develop" una vez que estaban listos y probados. Esto permite un desarrollo paralelo de características sin afectar la rama principal del proyecto.

Visual Studio Code & Git

Visual Studio Code proporciona una integración perfecta con Git, que te permite realizar tareas comunes de control de versiones directamente desde la interfaz del editor.

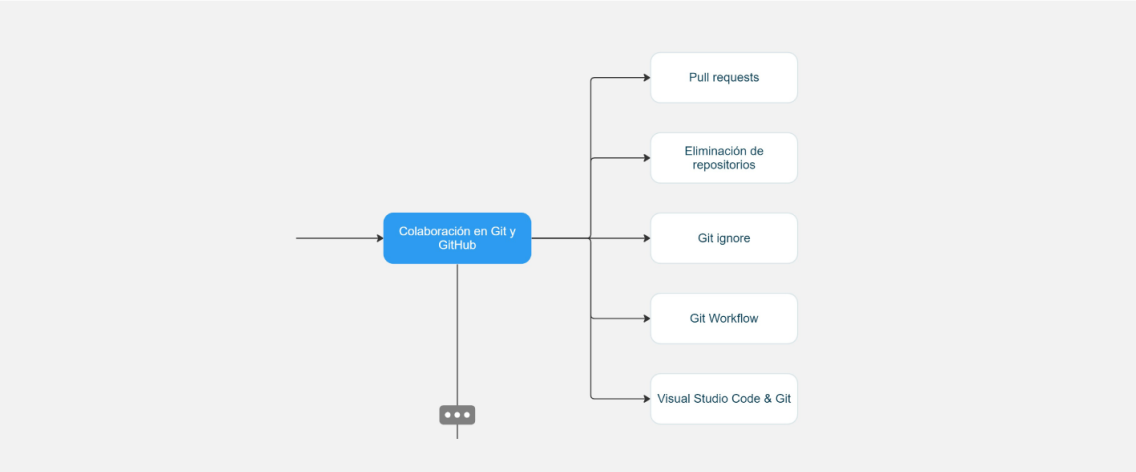
Esta integración agiliza el flujo de trabajo de desarrollo y facilita la colaboración con otros en tus proyectos.

Para profundizar en el funcionamiento de Git en Visual Studio Code, te invitamos a ver el siguiente video explicativo:

Mapa de conceptos vistos

¡Bien hecho! 🙌 Hemos llegado al final de nuestra clase sobre la colaboración en Git y GitHub.

En el día de hoy, exploramos la colaboración en Git y GitHub, abordando conceptos como los Pull Requests, el flujo de trabajo de ramas y la importancia de trabajar en equipo, además de lograr resolver los desafíos y ejercicios que te fuimos presentando.



Ahora tienes las herramientas necesarias para seguir adelante en tu viaje de desarrollo con confianza y eficiencia.

¡Sigue construyendo tu conocimiento y disfruta del proceso!

Hasta la próxima👋