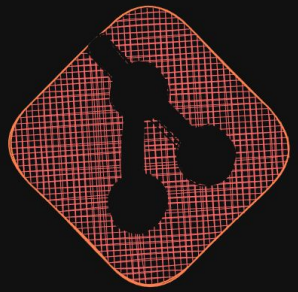
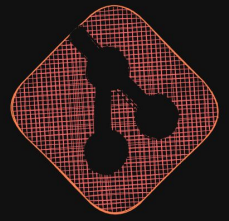
Git y GitHub

Clase 1:



- ¿Qué es Git y GitHub?
- Diferencias y beneficios de ambas
- Concepto de Commit
- Anatomía de Git
- Concepto de rama
- Trabajo con múltiples ramas
- Comandos básicos de Window en la terminal de Git

José Rafael Pérez Rivero joserafaelperezr@icloud.com



¿Qué es Git?

Git es un sistema de control de versiones distribuido que funciona localmente en tu computadora. Su propósito principal es gestionar cambios en archivos a lo largo del tiempo, registrando quién, cuándo y qué modificaciones se hicieron. Fue creado por Linus Torvalds

- Funciona como un "historial inteligente": Registra cada modificación en tus archivos (código, texto, imágenes, etc.), permitiéndote volver a versiones anteriores, comparar cambios y colaborar sin perder el trabajo previo.
- Es local y descentralizado: Cada persona tiene una copia completa del historial en su computadora, lo que permite trabajar sin conexión.

Un poco de historia de Git con Linus Torvalds

En el año 2005 Linus Torvalds, **el creador de Linux**, estaba en una pelea épica con los sistemas de control de versiones existentes. BitKeeper, el software que usaban para gestionar el código de Linux, les quitó la versión gratuita y, como buen programador enojado, Linus dijo:

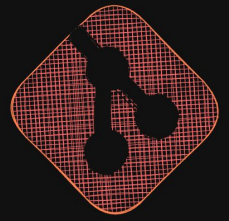
"¿Ah sí? Pues hago mi propio sistema de control de versiones... sin restricciones!"

Y así en menos de un mes Linus creó Git.

En la actualidad Git se convirtió en una herramienta esencial para cualquier desarrollador. **No hay empresa de que maneje software que no te vaya a pedir Git y GitHub** como requisitos. Linus, sin quererlo, **cambió el mundo del desarrollo**, todo por estar molesto con un sistema de control de versiones que no le gustaba.

Moraleja: Cuando algo no te guste, créalo tú mismo.

José Rafael Pérez Rivero joserafaelperezr@icloud.com



¿Git solo es para programadores?

Nada más lejos de la realidad. Aunque Git fue diseñado originalmente para proyectos de programación, su utilidad va mucho más allá. **Es una potente herramienta de control de versiones** que puede aplicarse a distintos campos, permitiendo gestionar cambios de manera ordenada y eficiente.

Usos de Git fuera de la programación:

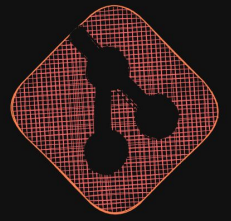
- **Escritores**: Gestionar versiones de un libro, tesis o artículo sin perder borradores anteriores.
- **Diseñadores**: Rastrear cambios en archivos de Photoshop, Figma o ilustraciones digitales.
- **Académicos**: Registrar cambios en estudios, documentos científicos o conjuntos de datos para garantizar la trazabilidad.
- **Trabajo con Office**: Controlar versiones de documentos de Word, Excel y PowerPoint, ideal para reportes, análisis de datos o presentaciones sin depender de nombres de archivos confusos como *"final_v2_DEFINITIVO.docx"*.

¿Qué es GitHub?

GitHub es una plataforma en la nube que aloja repositorios (proyectos) gestionados con Git. Puedes ponerlos públicos o privados. Añade herramientas para colaboración remota y funciones sociales (como comentarios, seguimiento de tareas o integración con otras apps).

Funcionalidades clave:

- Almacena tu proyecto en la nube (accesible desde cualquier lugar con todo tu historial de commits).
- Permite trabajar en equipo con herramientas como Pull Requests (solicitudes de cambios) o Issues (seguimiento de problemas).
- Ofrece integración con servicios como Slack, Google Drive, o incluso automatizaciones (GitHub Actions).



¿Git y GitHub son lo mismo?

No, Git y GitHub no son lo mismo, aunque están estrechamente relacionados.

- Git es un sistema de control de versiones que permite gestionar cambios en archivos de manera local.

- **GitHub**, en cambio, es una **plataforma en la nube** que utiliza Git para almacenar repositorios y facilitar la colaboración.

Analogía:
Git es como tener un cuaderno de notas en casa, donde escribes y editas a tu ritmo.

GitHub es como una biblioteca digital donde subes ese cuaderno para que otras personas lo consulten o contribuyan.

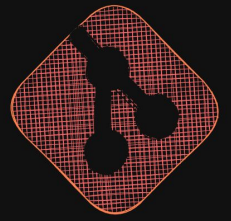
Puedes usar Git sin GitHub, pero GitHub necesita Git para funcionar.

Beneficios de usar Git y GitHub

Al combinar ambas herramientas, obtienes lo mejor de ambos mundos:

1. **Trabajo local y respaldo en la nube**:
   - Git te permite trabajar offline, y GitHub sirve como copia de seguridad remota.
   - Ejemplo: Si tu computadora se daña, tu proyecto sigue seguro en GitHub y puedes recuperar totalmente lo que tenías en tu proyecto simplemente accediendo desde otra computadora.

2. **Colaboración avanzada**:
   - Git para gestionar cambios y GitHub para revisar propuestas de equipo (**Pull Requests**), asignar tareas (**Issues**) o discutir mejoras.
   - Ejemplo: Un equipo de periodistas escriben un artículo conjunto y usan GitHub para coordinar quién edita cada sección.

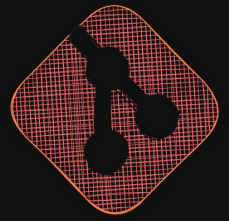


Beneficios de usar Git y GitHub

Al combinar ambas herramientas, obtienes lo mejor de ambos mundos:

3. **Automatización y flujos de trabajo**:
   - GitHub permite crear automatizaciones (GitHub Actions) para tareas repetitivas.
   - Ejemplo: Al subir un nuevo diseño, se genera automáticamente un PDF y se envía por correo al cliente.

4. **Transparencia y auditoría**:
   - El historial de Git + la accesibilidad de GitHub permiten verificar quién hizo cada cambio y cuándo.
   - Ejemplo: En un contrato legal, se puede rastrear quién modificó una cláusula específica.



Beneficios de usar Git y GitHub

Al combinar ambas herramientas, obtienes lo mejor de ambos mundos:

5. **Comunidad y portafolio**:
   - GitHub sirve como vitrina para mostrar proyectos (incluso no técnicos).
   - Ejemplo: Un escritor puede publicar su libro en progreso y recibir feedback de lectores.

Concepto de commit

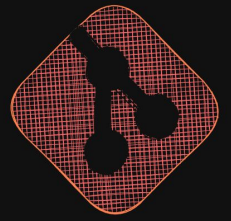
Un commit en Git es como una foto instantánea de el proyecto en un punto específico del tiempo. Cada vez que realizas cambios y los registras con un commit, Git guarda un estado del código para que puedas **volver atrás si quieres.**

Cada commit en Git tiene varios elementos clave que lo hacen único y rastreable en el historial del proyecto:

1. Hash único
   - Cada commit tiene un identificador alfanumérico único (SHA-1), que permite diferenciarlo de otros commits.

2. Autor
   - Registra quién realizó el commit, incluyendo el nombre y el correo electrónico configurado en Git.

José Rafael Pérez Rivero joserafaelperezr@icloud.com



Concepto de commit

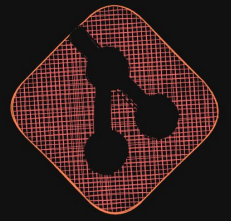
Cada commit en Git tiene varios elementos clave que lo hacen único y rastreable en el historial del proyecto:

3. Fecha y hora
   - Guarda el momento exacto en que se realizó el commit, lo que ayuda a rastrear la evolución del proyecto.

4. Mensaje del commit
   - Una breve descripción del cambio realizado, agregada por el usuario.

5. Referencia a commits anteriores
   - Cada commit, excepto el inicial, tiene un enlace al commit anterior, formando una cadena de historial.

José Rafael Pérez Rivero joserafaelperezr@icloud.com



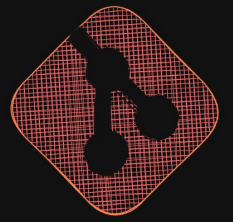
Concepto de commit

Cada commit en Git tiene varios elementos clave que lo hacen único y rastreable en el historial del proyecto:

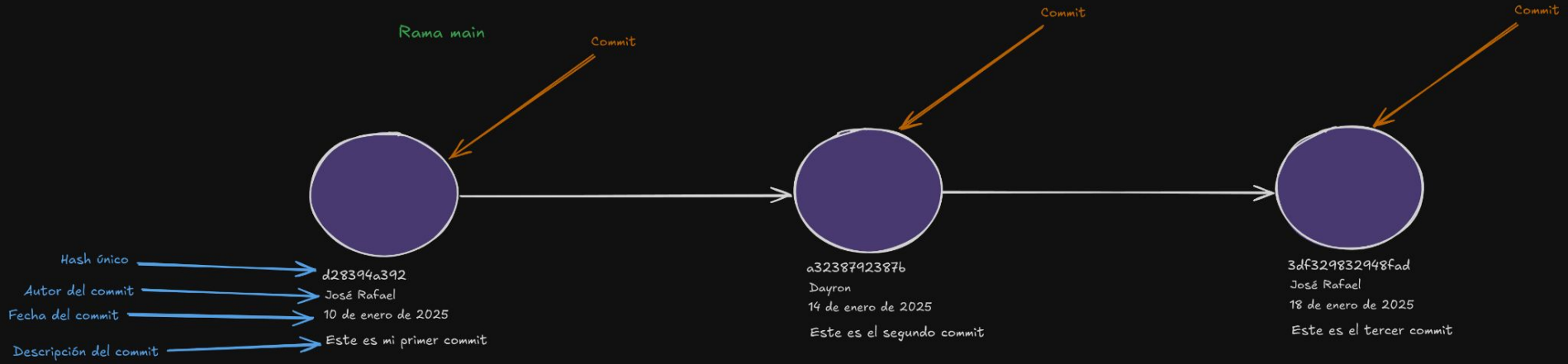
6. Contenido modificado
   - Guarda los cambios en los archivos afectados, permitiendo compararlos con versiones previas.

7. Estado de las ramas
   - Registra en qué rama se hizo el commit, lo que facilita la organización del código.

José Rafael Pérez Rivero joserafaelperezr@icloud.com



Ejemplo de lo anterior



Anatomía de Git

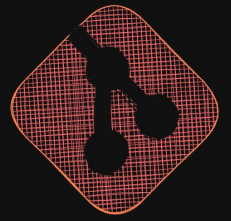
Git tiene 3 estados

Working Directory: Es donde editas archivos y realizas cambios. En este estado, Git aún no ha rastreado las modificaciones. Aquí es donde todo comienza.

Staging Area: Una zona intermedia donde agregas archivos antes de confirmarlos. Es decir, seleccionas cuáles cambios quieres guardar en la próxima "foto" (commit) del proyecto. Esto permite organizar los cambios antes de hacer un commit y evitar que se incluyan modificaciones no deseadas.

Git Repository: Donde los cambios quedan registrados en el historial del proyecto. En este estado git toma una “foto” (hace un commit) de los archivos que estaban en la zona de staging

Concepto de rama



Una rama (branch) en Git es una línea de desarrollo separada dentro de un repositorio. Es como una copia del código en la que puedes hacer cambios sin afectar la versión original.

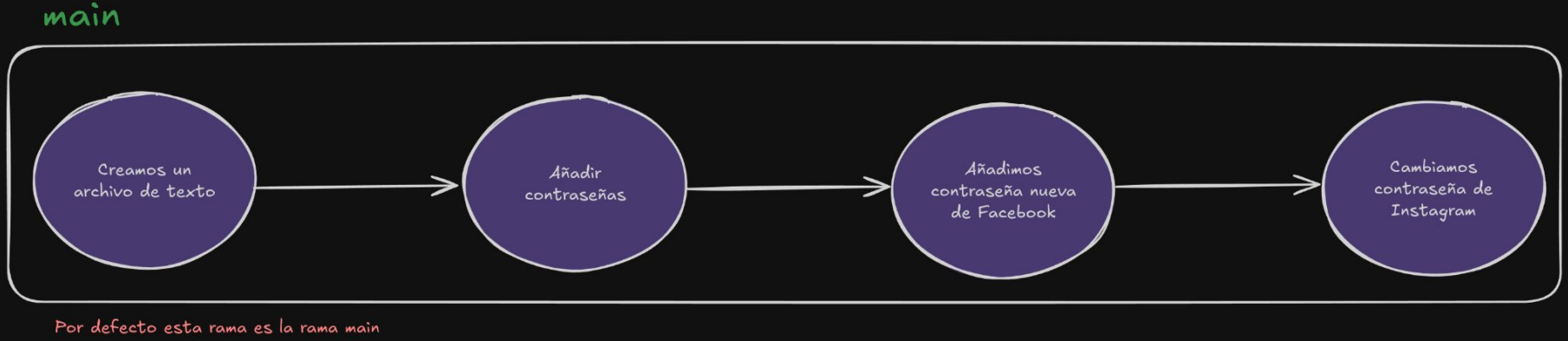
Desde el principio, cuando creas un repositorio Git, **la primera rama que se genera es main**. Esta rama suele ser la más importante, ya que representa la versión estable del proyecto, la base sobre la que se construyen nuevas funciones y correcciones.

Dato curioso:
En 2020, Git decidió cambiar el nombre predeterminado de la rama principal de master a main. Esto fue parte de un esfuerzo por adoptar un lenguaje más inclusivo, ya que el término "master" podía interpretarse como una referencia a la esclavitud. Este cambio fue bien recibido por la comunidad tecnológica.

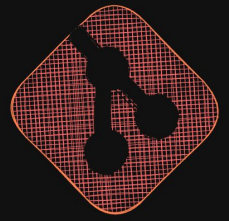
Concepto de rama

Desde el principio, cuando creas un repositorio Git, **la primera rama que se genera es main**. Esta rama suele ser la más importante, ya que representa la versión estable del proyecto, la base sobre la que se construyen nuevas funciones y correcciones.

Ejemplo visual:



¿Cómo funcionan las ramas?

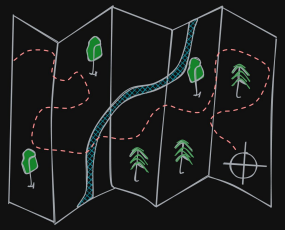


- main es la rama principal: Es el punto de partida y la versión estable del código.

- Puedes crear otras ramas: Desarrollas nuevas funciones sin tocar main.

- Fusionas ramas cuando los cambios están listos: Si una nueva funcionalidad funciona bien, la unes con main.

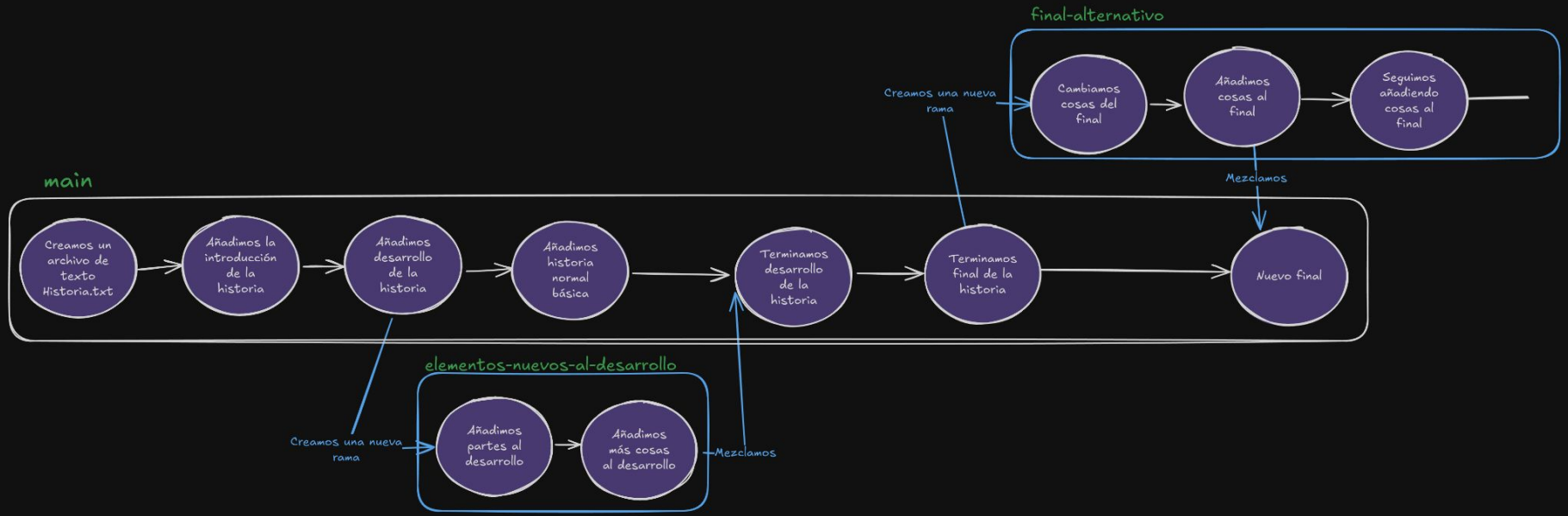
Ejemplo práctico:
Imagina que main es la receta original de un platillo. Si quieres probar una nueva versión con ingredientes diferentes, creas una rama aparte, la modificas y, cuando el resultado es bueno, actualizas la receta original (fusionas la rama con main).



¿Cómo funcionan las ramas?

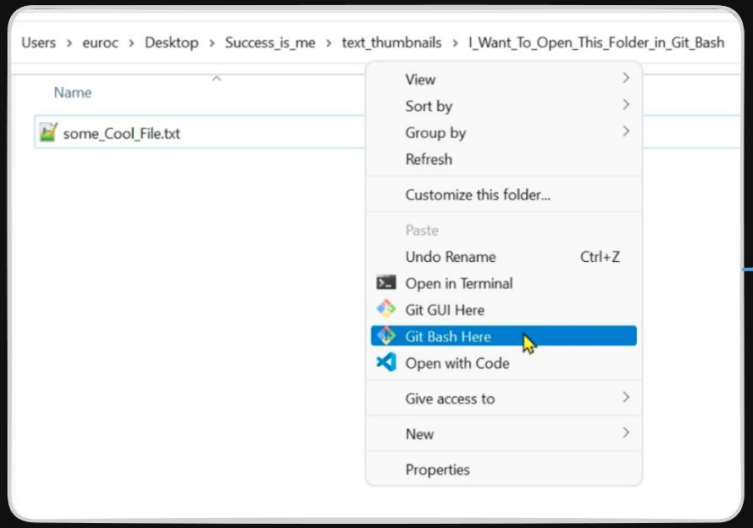
Ejemplo visual:

Imagina que estás escribiendo un documento de texto llamado "historia.txt", donde cuentas una historia.



Un poco de teoría de la terminal

Una vez instalado Git (explicación para Windows) abrimos la terminal que nos ofrece Git, de esta manera



Entonces se abrirá esta terminal

José Rafael Pérez Rivero joserafaelperezr@icloud.com

Algunos comandos básicos para la terminal (Windows)

- ls -> viene de “list”. Muestra todos los elementos que hay en la carpeta donde te encuentras.

- exit -> Sale de la terminal.

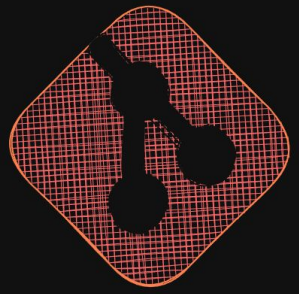
- cd -> Viene de “Change Directory”. Permite cambiar de carpeta.
  - cd <nombre-de-carpeta> -> Entra en la carpeta especificada.
  - cd .. -> Vuelve a la carpeta anterior.

- mkdir <nombre-de-carpeta> -> Crear una carpeta nueva

- rmdir <nombre-de-carpeta> -> Borrar carpeta.

- rm -r <nombre-de-carpeta> -> Borrar carpeta y su contenido.

José Rafael Pérez Rivero joserafaelperezr@icloud.com



Gracias por tu atención

Esta clase tuvo un enfoque teórico, con el objetivo de comprender Git y GitHub desde sus fundamentos, evitando una visión superficial.

Este conocimiento nos permitirá, en la próxima clase práctica, actuar con seguridad y entender cada paso que realizamos.

Información de contacto

@joserafael0160

@jose_rafael016

@jose_rafael0160

José Rafael Pérez Rivero joserafaelperezr@icloud.com