IT Academy

by KIBERNUM

Programa: Desarrollo de Aplicaciones Front-End

-

Módulo 2: Fundamentos de Desarrollo Front-End

Aprendizaje esperado

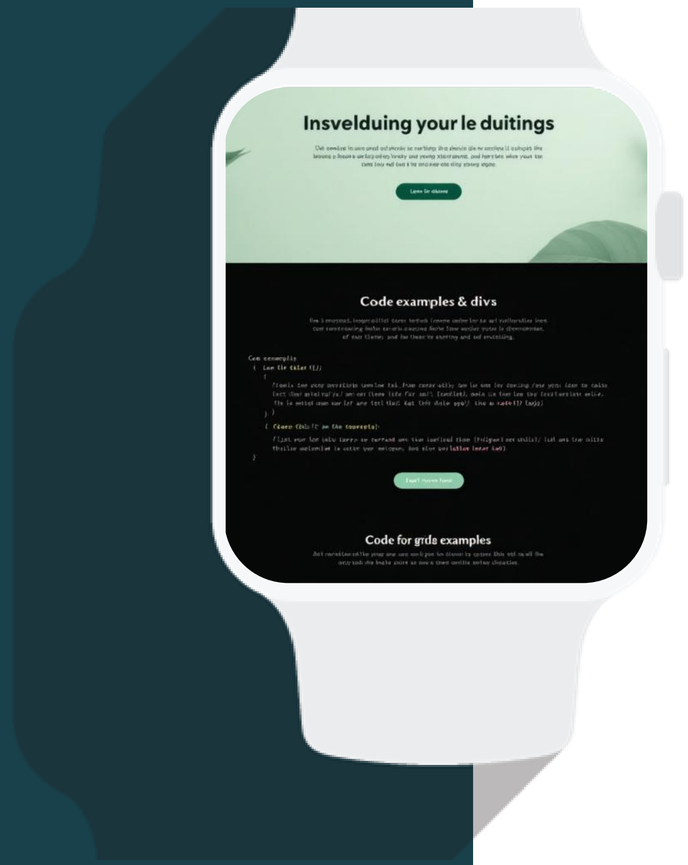
Gestionar el código fuente de un proyecto utilizando GitHub para mantener un repositorio de código remoto seguro y permitir trabajo concurrente.

¿Por qué es importante utilizar un sistema de control de versiones en el desarrollo Web?

Introducción a Git

Git es un sistema de control de versiones distribuido, creado por Linus Torvalds, el creador de Linux. Git es una herramienta esencial para gestionar el historial, la organización y la evolución del código en proyectos de software.

Beneficios de Git



1

Control de Versiones

Git permite navegar entre distintas versiones del código, lo que facilita la recuperación de versiones anteriores, la comparación de cambios y la identificación de errores.

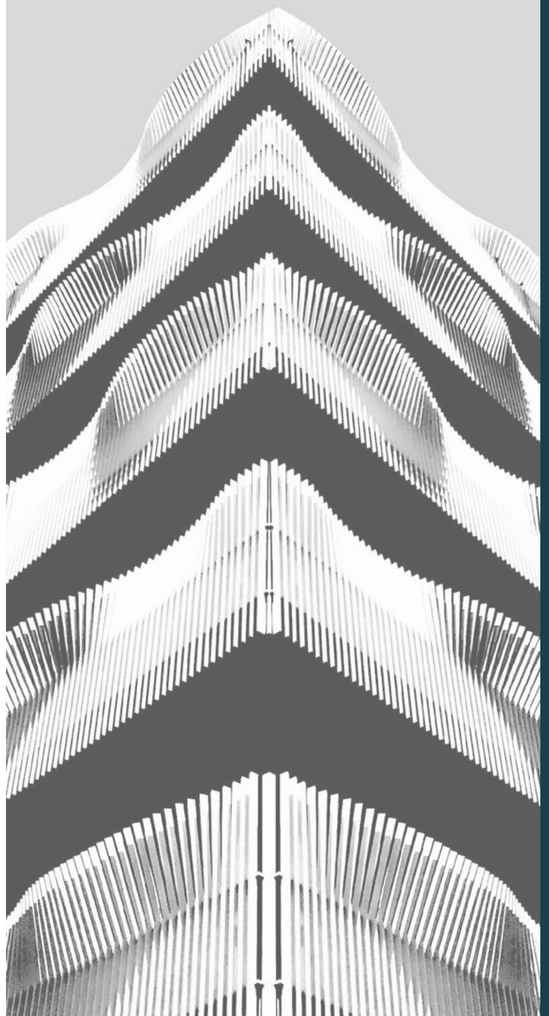
2

Trabajo en Equipo

Git facilita la colaboración entre desarrolladores, permitiendo que varios miembros trabajen en paralelo y sincronizar sus contribuciones de manera eficiente.

IT Academy
by KIBERNUM

Características Claves de Git



1

GESTIÓN DE RAMAS

Git permite crear y fusionar ramas de manera eficiente, lo que fomenta que los desarrolladores trabajen en nuevas funcionalidades en ramas separadas, facilitando la integración cuando los cambios estén listos.

2

DISTRIBUCIÓN DE CAMBIOS

Git opera como un sistema distribuido, lo que significa que cada desarrollador tiene una copia completa del repositorio. Los cambios se pueden importar como ramas adicionales y fusionar de la misma manera que se trabaja en la rama local.

3

EFICIENCIA EN PROYECTOS GRANDES

Git está optimizado para manejar grandes volúmenes de código de manera ágil, permitiendo una gestión fluida incluso en proyectos de gran escala.

4

REEMPAQUETADO PERIÓDICO

Git optimiza el almacenamiento de cambios mediante reempaquetado periódico, lo que mejora la eficiencia en el uso del espacio en disco y acelera las operaciones de recuperación.

Terminología Básica de Git

- **Commit**

Es el estado de un proyecto en un determinado momento de la historia del mismo. Generalmente va acompañado de un mensaje descriptivo con los cambios que contiene.

- **Repositorio**

Es la carpeta principal, donde se encuentran almacenados los archivos que componen el proyecto.

- **Rama (Branch)**

Es una línea alterna del tiempo, en la historia de nuestro repositorio. De esta forma no afectamos la versión estable del proyecto que es la rama principal. Por defecto, la rama principal es la rama máster.

El Concepto de Ramas en Git

- **Rama Principal**

La rama principal (master) contiene la versión estable del proyecto..

- **Rama de Desarrollo**

Las ramas de desarrollo se utilizan para trabajar en nuevas funcionalidades sin afectar la rama principal.

- **Fusión**

Cuando el trabajo en una rama de desarrollo está listo, se fusiona con la rama principal.

La Importancia de un Repositorio de Código Fuente

01.

Control de Cambios
Un repositorio de código fuente permite mantener un historial preciso de todas las versiones del proyecto, lo que facilita la identificación de errores y la recuperación de versiones anteriores.

02.

Colaboración
Un repositorio centralizado facilita la colaboración entre los desarrolladores, permitiendo que varios miembros trabajen en paralelo y sincronizar sus contribuciones de manera eficiente.

03.

Seguridad
Un repositorio de código fuente asegura que todas las modificaciones estén documentadas, se puedan rastrear y corregir, garantizando un desarrollo más organizado.

Introducción a Git Instalación y Configuración

A continuación, vamos a detallar los pasos necesarios para instalar y configurar Git, además de familiarizarnos con algunos comandos fundamentales que te permitirán comenzar a utilizar esta poderosa herramienta de control de versiones.

Instalación de Git

Windows

Descarga el instalador desde la página oficial de Git Downloads. Sigue las instrucciones paso a paso. Asegúrate de seleccionar la opción que incluye Git Bash, una terminal que facilita el uso de comandos Git en Windows.

macOS

Puedes instalar Git usando Homebrew con el comando: brew install git. Alternativamente, descárgalo directamente desde la página oficial y sigue las instrucciones.

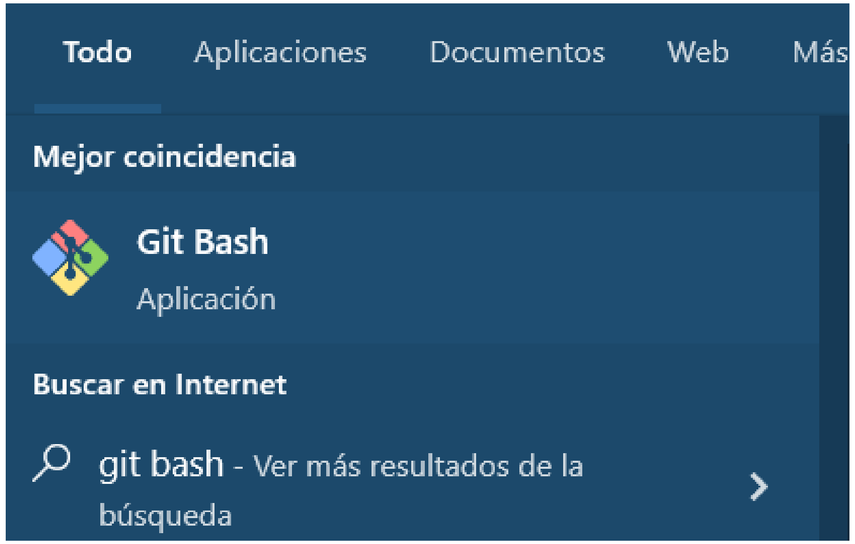
Linux

En distribuciones basadas en Debian (como Ubuntu), puedes instalar Git ejecutando: sudo apt install git. En distribuciones basadas en Red Hat, utiliza: sudo yum install git.

Configuración Inicial de Git

Abrir Git Bash

Haz clic en el botón de Inicio o presiona la tecla Windows en tu teclado. Escribe "Git Bash" en el campo de búsqueda. Haz clic en el icono de Git Bash cuando aparezca en los resultados de búsqueda.



Configuración Inicial de Git

Configurar Nombre y Correo Electrónico

Git utiliza esta información para identificar quién realiza los cambios en los repositorios.

Ejecuta los siguientes comandos en Git Bash:

```
$ git config --global user.name "Mario"
```

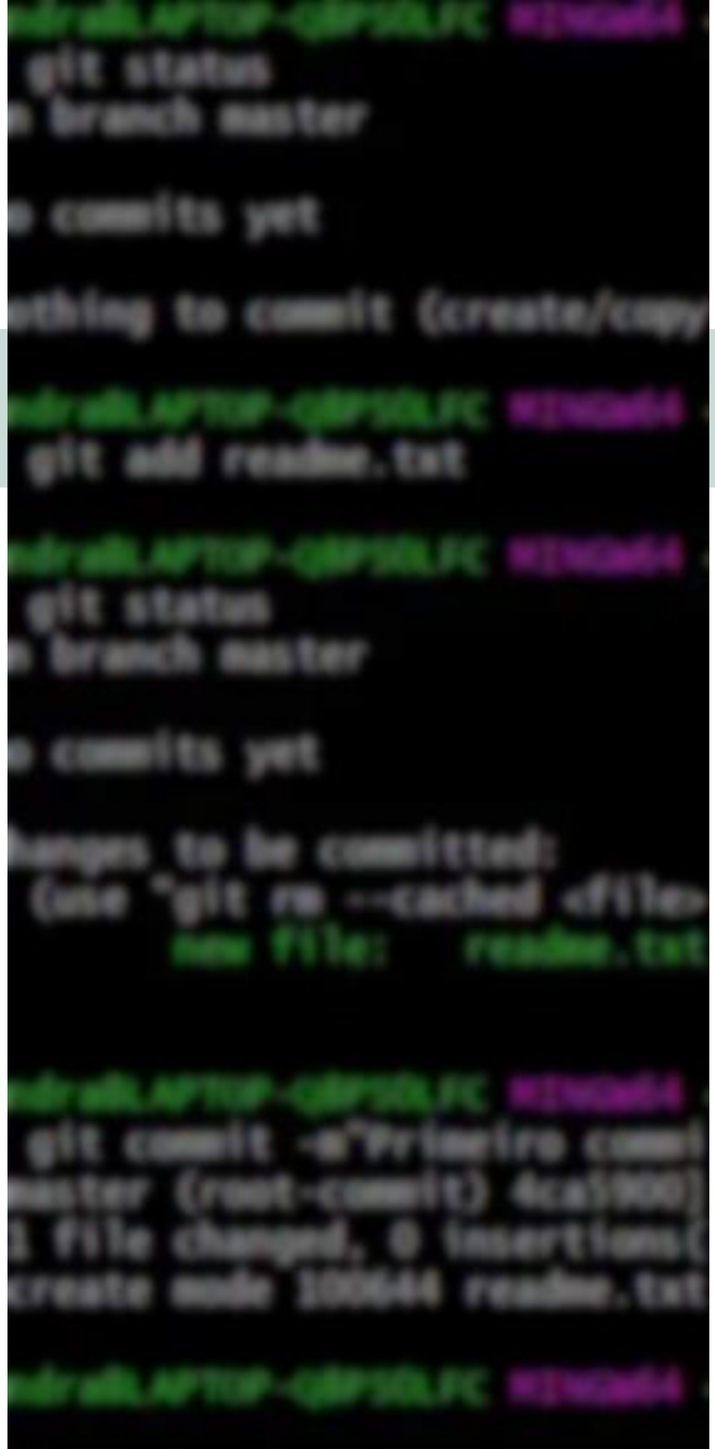
```
$ git config --global user.email "tuemail@example.com
```

Configuración Inicial de Git

Verificar Configuración

Puedes verificar que los datos se hayan configurado correctamente utilizando el siguiente comando:

```
$ git config --list
```

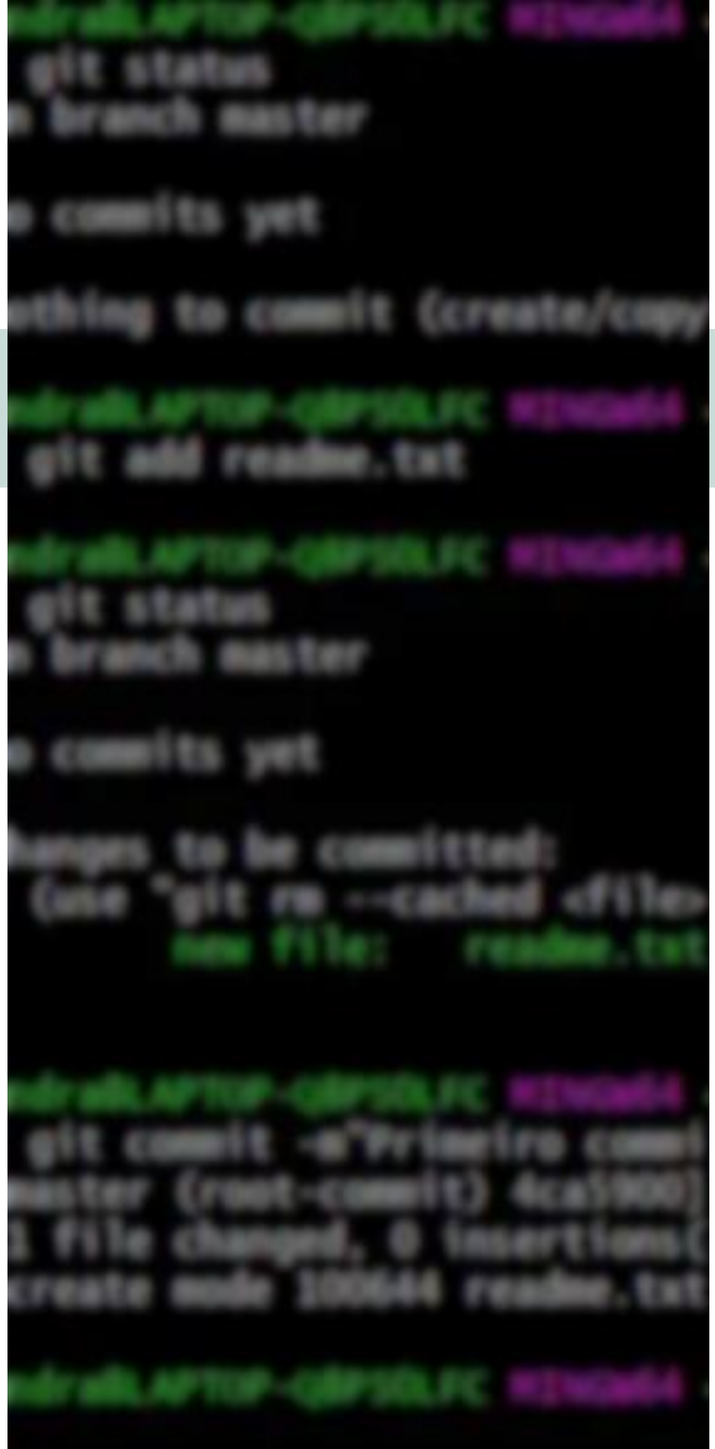


Comandos Básicos de Git

- **Inicializar un repositorio: git init**

Este comando crea un nuevo repositorio Git en el directorio actual. Inicializa una carpeta oculta llamada .git que contiene toda la información necesaria para el control de versiones.

```
USUARIO@DESKTOP-5JHP2VJ MINGW64 ~/Desktop/nueva_carpeta
$ git init
Initialized empty Git repository in C:/Users/USUARIO/Desktop/nueva_carpeta/.git/
```

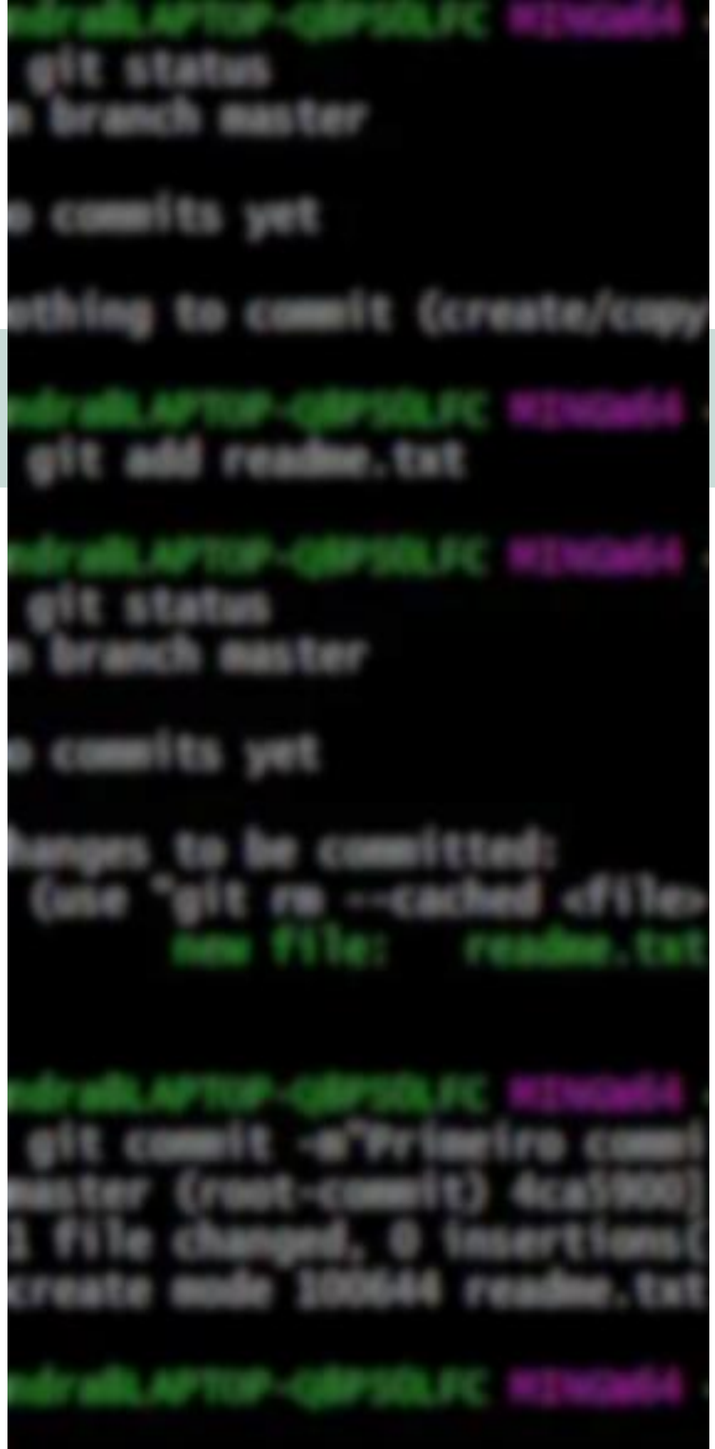


Comandos Básicos de Git

- **Clonar un repositorio: git clone <URL>**

Este comando copia un repositorio Git existente desde un servidor remoto (como GitHub) a tu máquina local. Es útil para comenzar a trabajar en un proyecto ya existente.

```
$ git clone <url-del-repositorio>
```

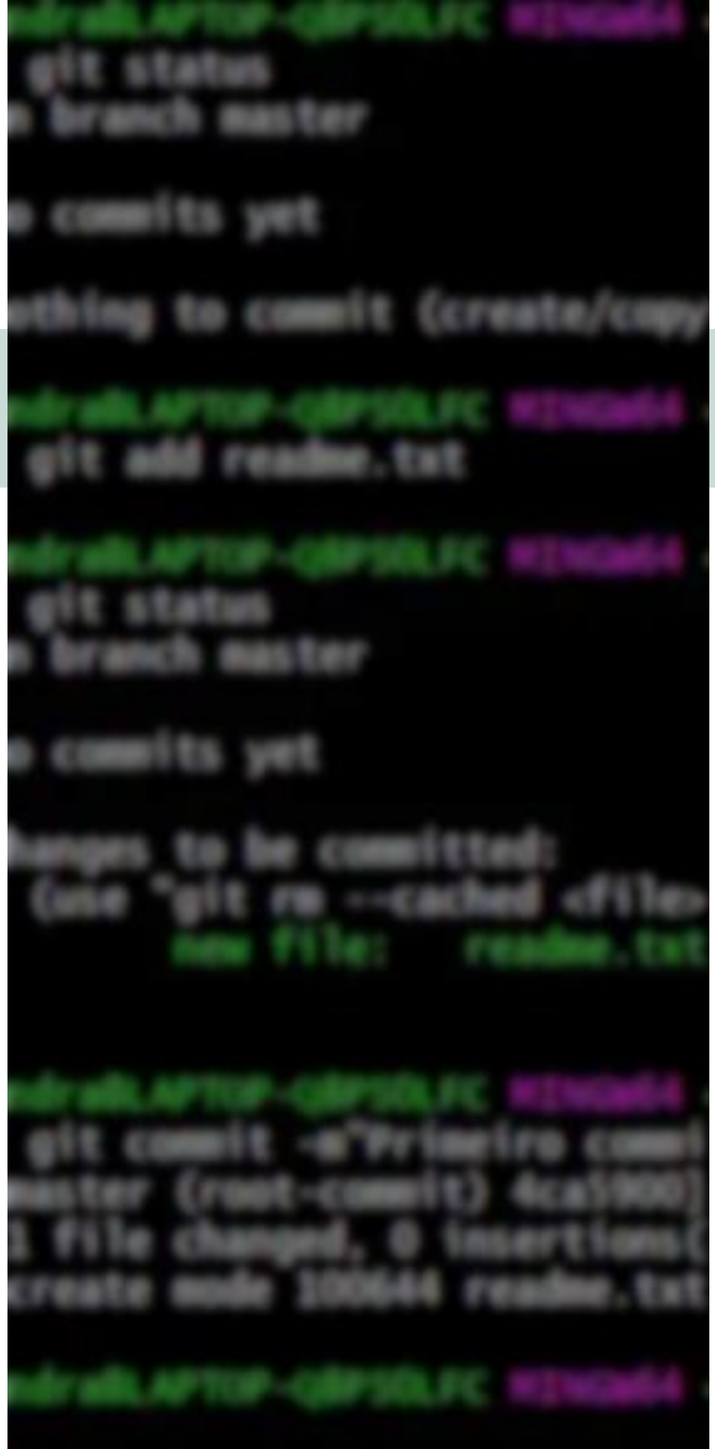


Comandos Básicos de Git

- **Agregar archivos: git add <archivo>**

Este comando agrega cambios en un archivo o varios al área de preparación (staging area). Los cambios en esta área están listos para ser confirmados en el repositorio.

```
USUARIO@DESKTOP-5JHP2VJ MINGW64 ~/Desktop/nueva_carpeta (master)
$ git add index.html
```

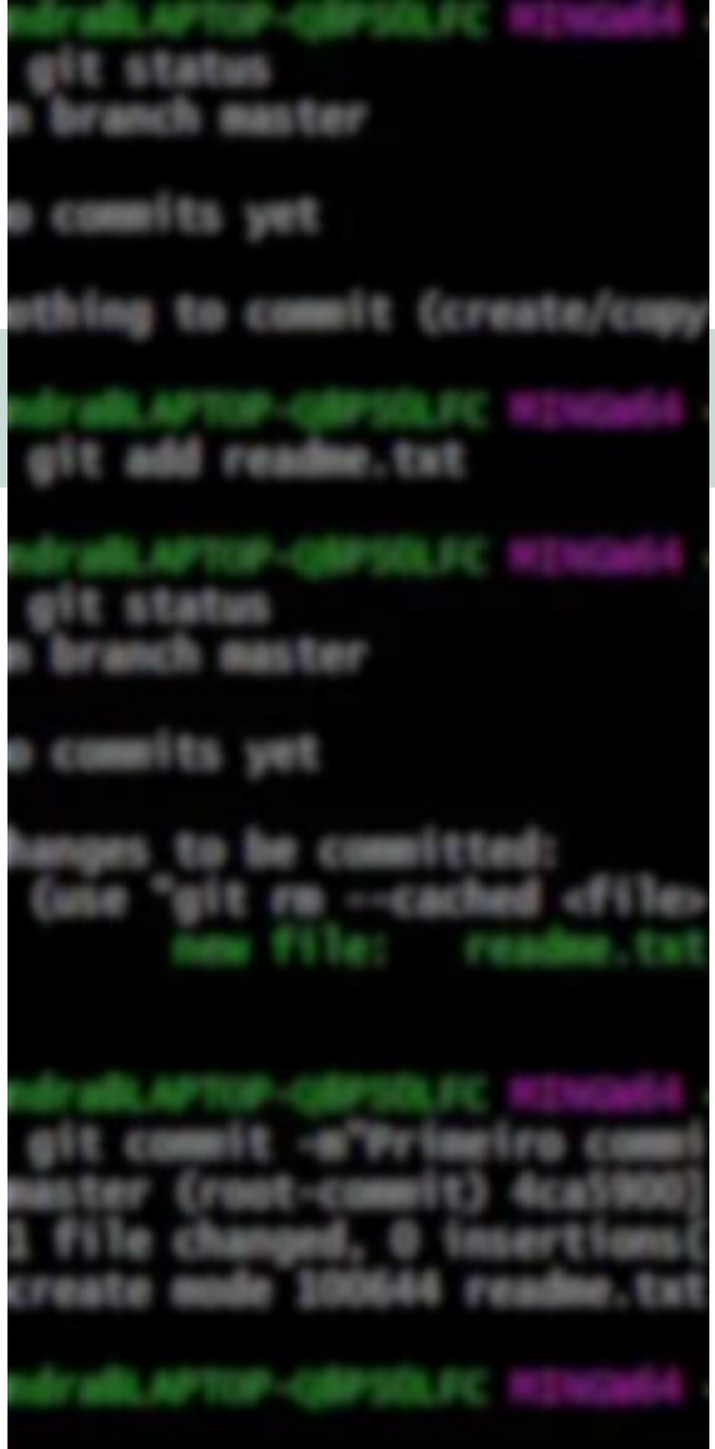


Comandos Básicos de Git

- **Confirmar los cambios (commit): git commit -m "Mensaje del commit"**

Este comando guarda los cambios que están en el área de preparación en el historial del repositorio. Cada commit es un punto de control que puedes recuperar en el futuro.

```
USUARIO@DESKTOP-5JHP2VJ MINGW64 ~/Desktop/nueva_carpeta (master)
$ git commit -m "mi primer commit"
```



Comandos Básicos de Git

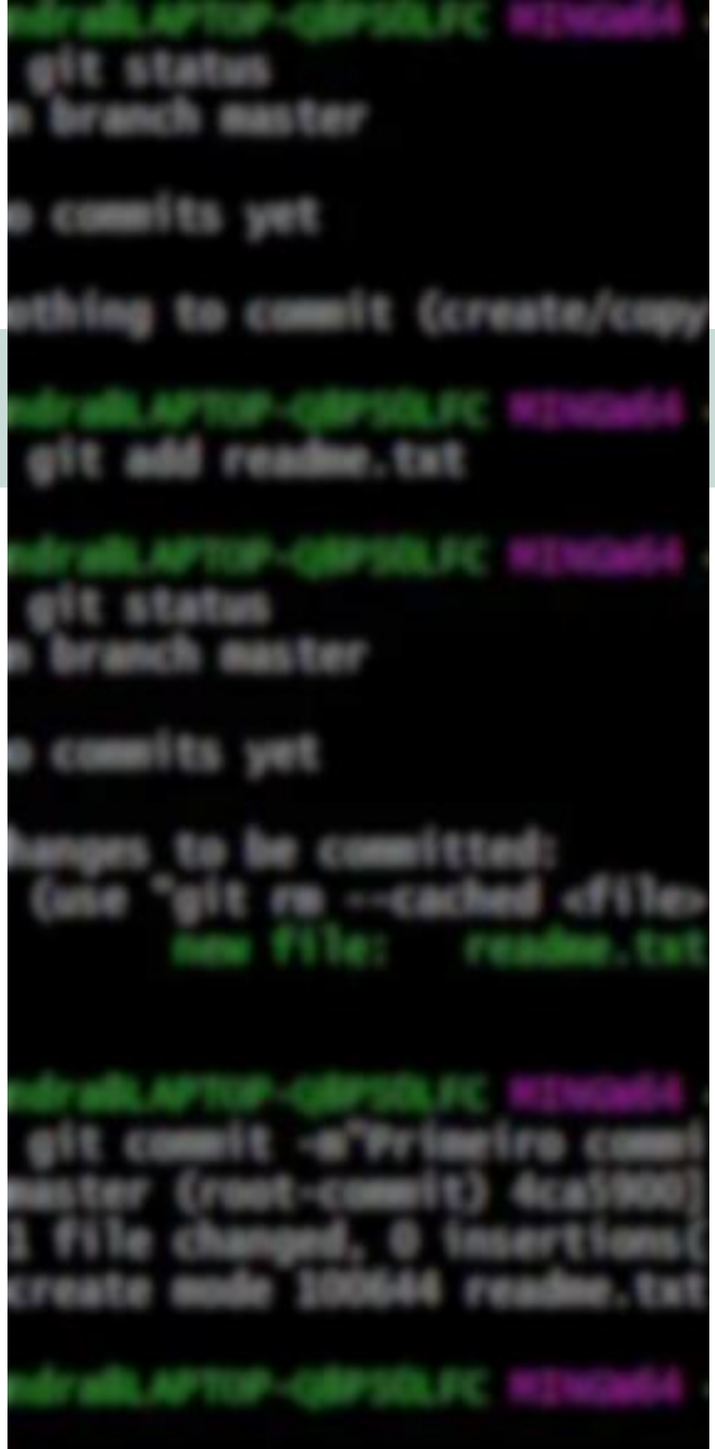
- **Ver el estado del repositorio: git status**

Este comando muestra el estado actual del repositorio, incluyendo archivos modificados, archivos en el área de preparación y archivos que no están siendo rastreados por Git.

```
USUARIO@DESKTOP-5JHP2VJ MINGW64 ~/Desktop/nueva_carpeta (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

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

Commits y Restauración de Archivos

Un commit en Git es un punto de control en la historia del proyecto. Registra un conjunto de cambios realizados en el código, lo que permite documentar el progreso y recuperar versiones anteriores si es necesario.



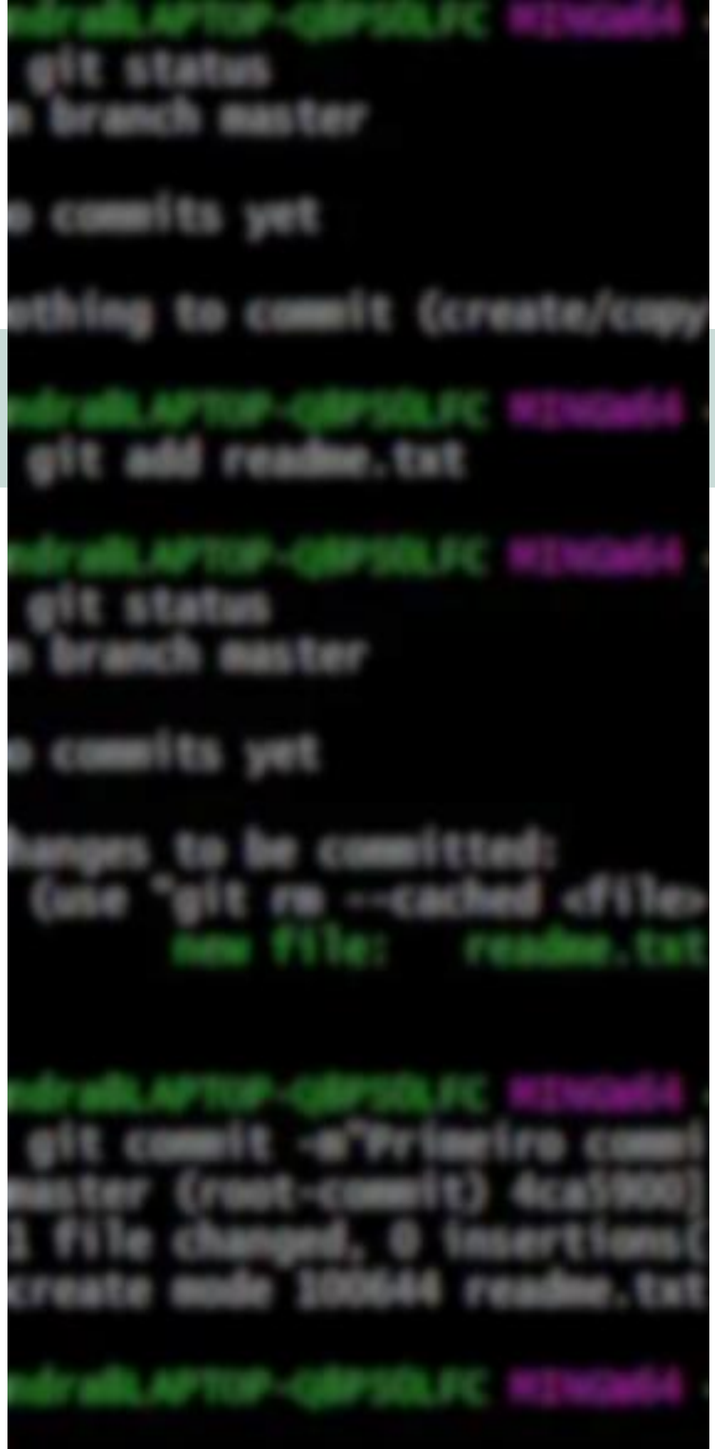
Realizar un Commit

- **Cómo realizar un Commit**

Para realizar un commit, primero debes agregar los archivos al área de preparación con el comando git add. Luego, puedes confirmar esos cambios con el siguiente comando:

```
@DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git commit -m "Descripción clara de los cambios realizados"
```

Este comando crea un nuevo punto de restauración en la historia del proyecto, permitiendo que los cambios sean registrados y versionados.



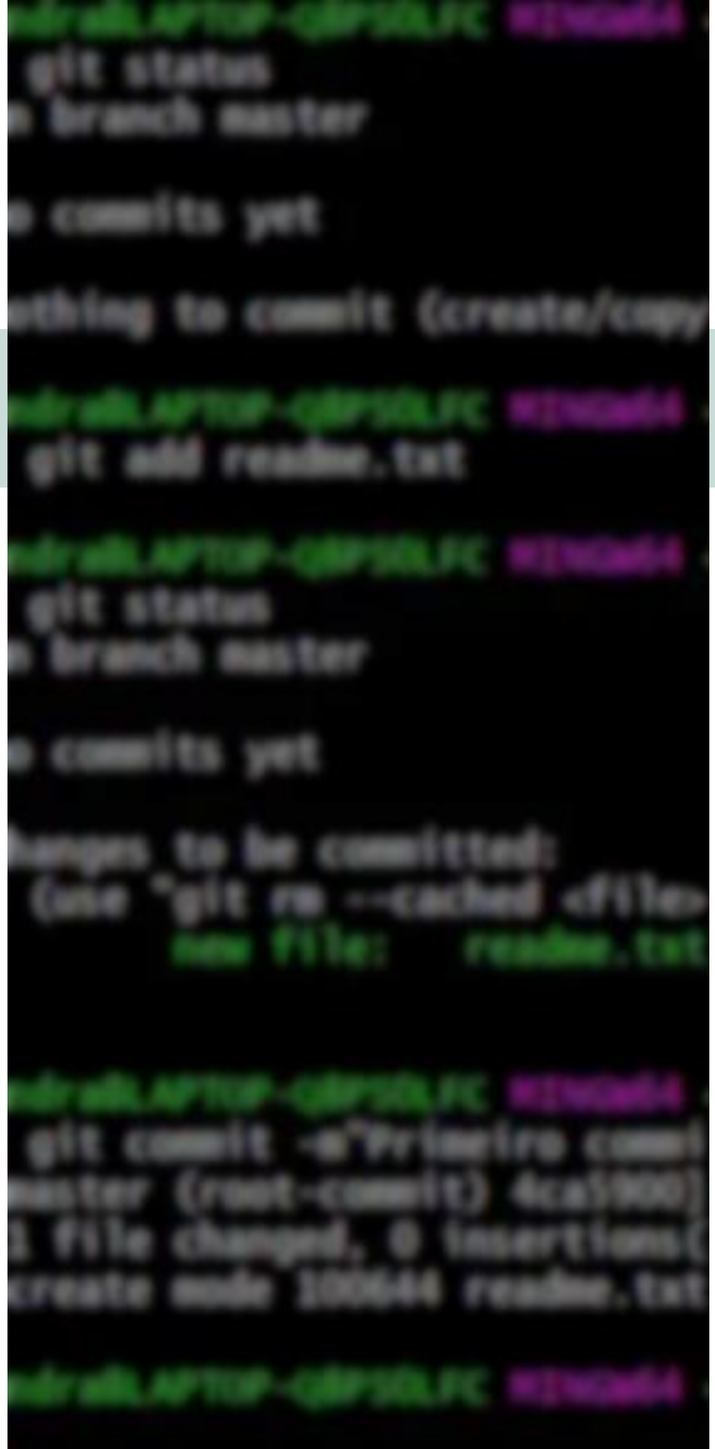
Restaurar Archivos a un Estado Anterior

- Si has realizado modificaciones no deseadas, Git permite restaurar un archivo a su estado anterior sin tener que confirmar los cambios.

- Para restaurar cambios no confirmados, usa el siguiente comando:

```
@DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git restore nombre-archivo
```

Este comando revertirá los cambios locales no confirmados y restaurará el archivo a su última versión confirmada.



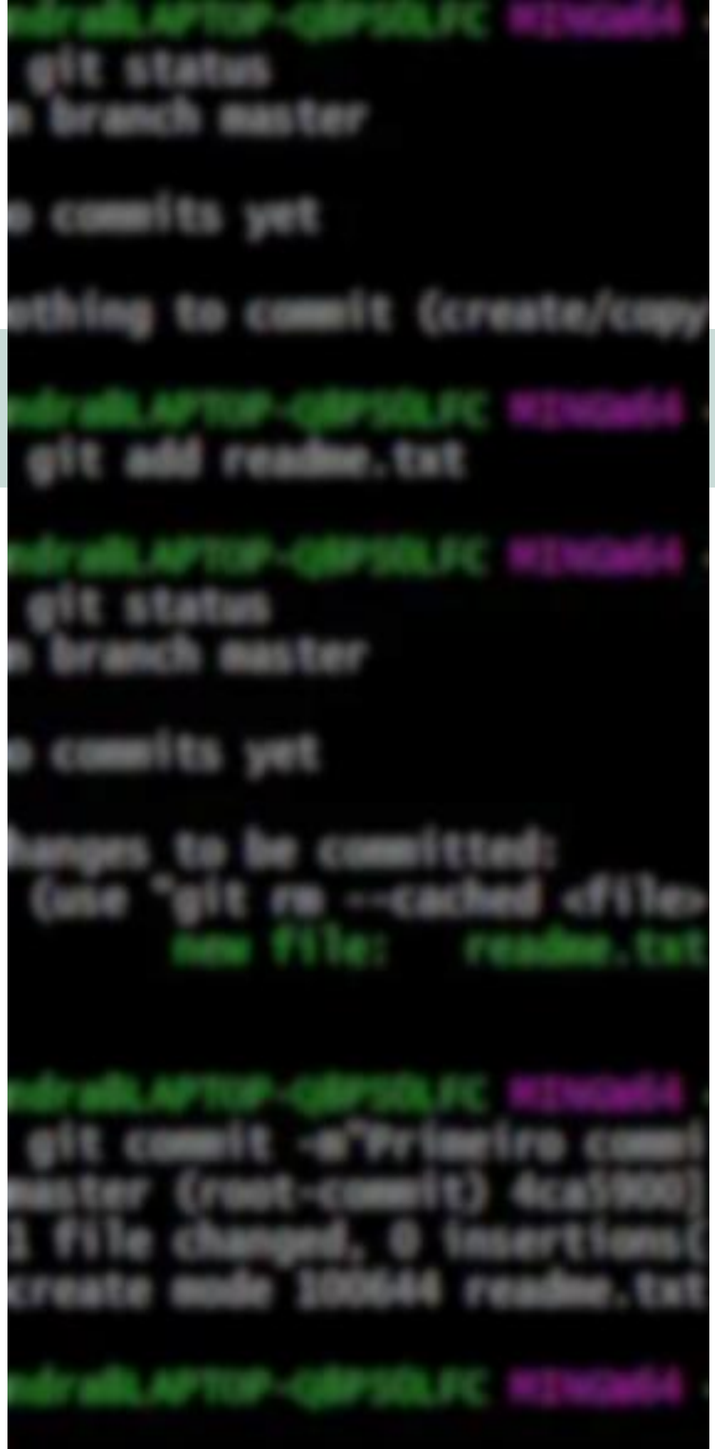
Cambios de Nombre de Archivos

- Si necesitas cambiar el nombre de un archivo en tu proyecto, Git ofrece el comando git mv. Este comando automáticamente renombra el archivo y lo mueve al área de preparación en un solo paso.

- Para renombrar un archivo:

```
@DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git mv nombre-antiguo nombre-nuevo
```

Esto evita la necesidad de eliminar manualmente el archivo antiguo y agregar el nuevo, facilitando el proceso de renombrado.

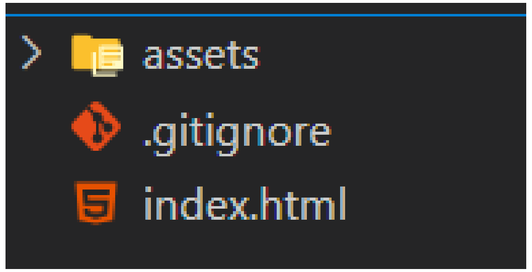


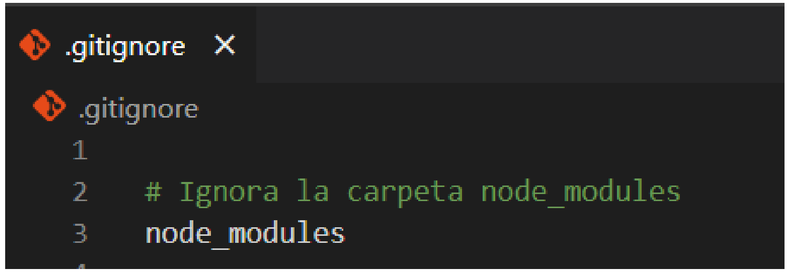
Ignorar Archivos

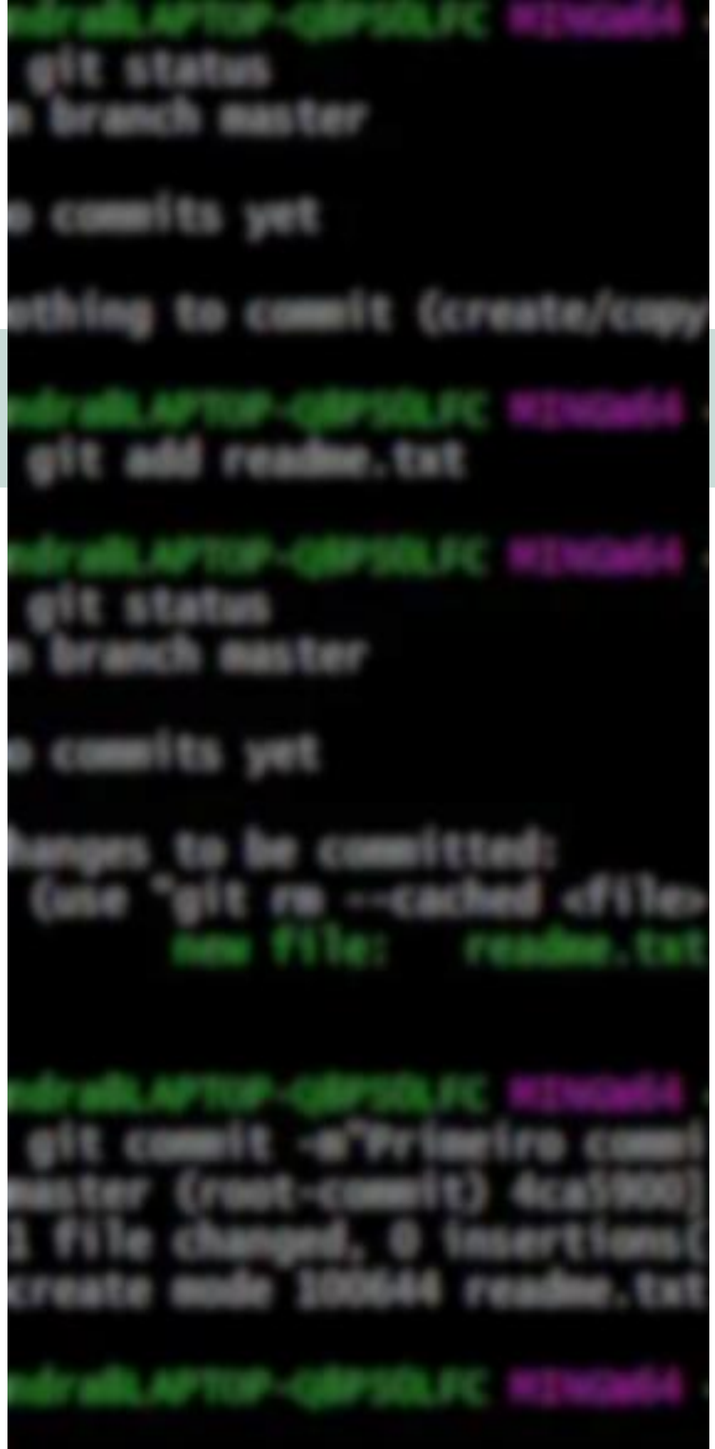
- En ocasiones, puede que no quieras que Git rastree ciertos archivos, como archivos de configuración local o directorios generados automáticamente. En estos casos, puedes usar el archivo .gitignore.

- Para crear o modificar .gitignore, abre o crea el archivo en la raíz de tu repositorio y agrega los nombres de los archivos o directorios que deseas excluir.

- Ejemplo:







Eliminar Archivos del Seguimiento

🔹 Si un archivo ya ha sido agregado previamente al repositorio y deseas que Git deje de rastrearlo, debes eliminarlo del seguimiento utilizando el comando:

```
git rm --cached <archivo>
```

🔹 Esto eliminará el archivo del repositorio, pero no lo borrará del sistema de archivos local.

Reto: Primer commit

🔹 **Instrucciones:**

🖥️ Inicializar un repositorio en su máquina local usando git init.

✍️ Crear un archivo (ejemplo: notas.txt) y agregar contenido.

📂 Agregar el archivo al área de preparación con git add notas.txt.

✅ Realizar un commit con un mensaje descriptivo usando git commit -m "Primer commit".

Ramas: Uniones, Conflictos y Tags

- Las ramas en Git permiten trabajar simultáneamente en diferentes funcionalidades sin interferir con la rama principal (usualmente llamada main o master).

- Esto ayuda a los desarrolladores a aislar su trabajo hasta que esté listo para ser integrado, facilitando la gestión de proyectos.

Ramas

Creando una nueva rama

Para crear una nueva rama, usa el siguiente comando:

```
.DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git branch rama_secundaria
```

Esto crea una nueva rama sin cambiar el contexto actual de tu trabajo.

Cambiando entre ramas

- Desde Git 2.23, el comando recomendado para cambiar entre ramas es git switch. Sin embargo, git checkout sigue siendo válido.
- Para cambiar de rama con git switch, utiliza:

```
.DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git switch rama_secundaria
```

Ramas

- **Unir ramas**

Una vez que termines de trabajar en una rama, puedes unir los cambios con otra rama utilizando el comando git merge.

```
@DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git merge main
```

Si existen conflictos entre ambas ramas, Git te pedirá que resuelvas estos conflictos manualmente antes de completar la unión.

Tags

- **Usando Tags para marcar puntos claves**

- Los tags son útiles para marcar puntos clave en la historia del proyecto, como el lanzamiento de una versión específica.
- Para crear un tag:

```
@DESKTOP-Q7LG5E8 MINGW64 ~/Desktop
$ git tag nevo_tag
```

Esto no afecta la historia del proyecto, pero te permite referenciar ese punto en el tiempo fácilmente.

IT Academy

by KIBERNUM