GIT FOR DUMMIES

GUÍA BÁSICA

Cristel Estefanía

Tabla de contenido

INTRODUCCIÓN

¡Bienvenidos a *GIT FOR DUMMIES*! Este manual está diseñado para ser una guía práctica y sencilla para entender y dominar GIT, una de las herramientas más importantes en el desarrollo de software, análisis de datos y gestión de proyectos.

GIT es un sistema de control de versiones que permite organizar, rastrear y colaborar en proyectos de manera eficiente. Ya sea que trabajes con scripts, análisis de datos, documentos o cualquier otro archivo, GIT te ayudará a mantener un registro de los cambios, probar nuevas ideas sin temor a cometer errores y colaborar con otros sin conflictos.

Git fue creado por Linus Torvalds en 2005 para el desarrollo del kernel de Linux. Su propósito es ser rápido, eficiente y permitir un manejo robusto de versiones. A diferencia de los sistemas de control de versiones centralizados, Git permite que cada desarrollador tenga una copia completa del repositorio.

Este manual está pensado para explicar los conceptos desde cero, con ejemplos claros y prácticos que pueden aplicarse tanto en proyectos académicos como profesionales. No es necesario ser experto en programación; GIT es una herramienta que cualquiera puede aprender y aprovechar para organizar mejor su trabajo.

Si estás buscando una forma de optimizar tus proyectos y mantener todo bajo control, este manual será tu mejor aliado.

CONCEPTOS BÁSICOS

Repositorio

Un repositorio es un espacio de almacenamiento donde se guarda el código fuente y el historial de un proyecto. Contiene todos los archivos, así como la información sobre las versiones y cambios realizados a lo largo del tiempo.

- Repositorio local: es la copia del repositorio que se encuentra en tu máquina. Puedes trabajar en él sin conexión a Internet y realizar cambios de manera local. Los commits se realizan en este repositorio antes de ser enviados al remoto.

- Repositorio remoto: es una versión del repositorio que se encuentra en un servidor en la nube (como GitHub, GitLab, etc.). Permite la colaboración entre diferentes desarrolladores, ya que puedes enviar (push) y recibir (pull) cambios desde este repositorio.

Commits

Un **commit** es una acción que guarda un "estado" del proyecto en el repositorio. Cada commit incluye un mensaje descriptivo y un identificador único, lo que permite rastrear los cambios realizados. Al realizar un commit, se guarda la versión actual de los archivos y se registra en el historial del proyecto, lo que facilita la recuperación de versiones anteriores si es necesario.

Branch (Rama)

Una **rama** es una versión paralela del código que permite trabajar en diferentes características o correcciones sin afectar el código principal (generalmente la rama main o master). Las ramas son útiles porque permiten:

- Desarrollar nuevas funcionalidades de manera aislada.
- Realizar correcciones de errores sin interrumpir el flujo de trabajo principal.

- Facilitar la colaboración entre varios desarrolladores, cada uno trabajando en su propia rama.

Clone y Fork

Clone: es una operación que crea una copia local del repositorio remoto. Al clonar un repositorio, obtienes todos los archivos y el historial de cambios. Esto te permite trabajar en el proyecto de forma local.

Fork: es una copia del repositorio que se encuentra en un servidor remoto, pero es gestionada por un usuario diferente. Se utiliza principalmente en plataformas como GitHub para permitir a los usuarios hacer modificaciones a un proyecto sin afectar el repositorio original. Una vez que se realizan los cambios en un fork, el autor puede enviar una solicitud de incorporación (pull request) para que los mantenedores del repositorio original consideren integrar sus cambios.

INSTALACIÓN Y CONFIGURACIÓN

Cómo instalar GIT en diferentes sistemas operativos (Windows, MacOS, Linux).

Configuración inicial:

git config --global user.name "Tu Nombre"

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

COMANDOS BÁSICOS DE GIT

Iniciar un repositorio:

git init

Añadir archivos al área de preparación:

git add archivo.txt

Guardar cambios (Commit):

git commit -m "Descripción del cambio"

Ver el estado del repositorio:

git status

Clonar un repositorio existente:

git clone https://github.com/usuario/repo.git

Enviar cambios al repositorio remoto.:

git push

Actualizar tu repositorio local con cambios del remoto:

git pull

RAMAS

Las ramas permiten desarrollar características o corregir errores sin afectar el código principal.

Merge

El **merge** es el proceso de fusionar dos ramas. Cuando un desarrollador ha terminado de trabajar en una característica en su rama, puede usar el merge para combinar esos cambios en la rama principal o en otra rama. Esto permite integrar el trabajo de diferentes colaboradores y mantener el proyecto actualizado. El merge puede ser automático, pero, a veces, puede requerir resolución de conflictos si hay cambios contradictorios en las ramas.

Crear una rama:

git branch nueva-rama

Cambia a la nueva rama:

git checkout nueva-rama

Realizar el merge:

git merge nueva-rama

COLABORACIONES CON GIT

Uno de los mayores beneficios de usar GIT es la capacidad de trabajar en equipo de forma organizada y eficiente, incluso en proyectos complejos. GIT permite que varias personas trabajen simultáneamente en diferentes partes de un proyecto sin sobrescribir ni perder el trabajo de otros. Para lograr esto, es fundamental comprender cómo colaborar utilizando repositorios remotos.

1. Configuración de un repositorio remoto

Para trabajar en colaboración, es necesario conectar el repositorio local con uno remoto, como los que ofrece plataformas como GitHub, GitLab o Bitbucket. Esto se hace utilizando el comando:

git remote add origin URL-del-repositorio

La URL del repositorio es el enlace que se obtiene al crear un repositorio en la plataforma remota.

2. Subir cambios al repositorio remoto

Para compartir el trabajo con el equipo, los cambios locales deben subirse al repositorio remoto:

git push origin nombre-rama

Por ejemplo, para subir los cambios de una rama llamada main:

git push origin main

3. Obtener cambios del repositorio remoto

Antes de comenzar a trabajar, es importante asegurarse de que el repositorio local esté actualizado con los últimos cambios realizados por el equipo:

git pull origin nombre-rama

4. Flujo típico de colaboración

1. Clonar el repositorio remoto:

git clone URL-del-repositorio

2. Crear una rama para trabajar en una nueva funcionalidad o corrección:

git branch nueva-rama

git checkout nueva-rama

3. Hacer cambios, agregarlos al área de preparación y realizar commits.
4. Subir la rama al repositorio remoto:

git push origin nueva-rama

5. Realizar un **Pull Request** (en GitHub) para que otros revisen e integren los cambios a la rama principal.

RESOLUCIÓN DE CONFLICTOS

Los conflictos en GIT ocurren cuando dos o más personas realizan cambios en las mismas líneas de un archivo o cuando un cambio en una rama choca con el de otra rama durante un merge. Aunque los conflictos pueden parecer intimidantes al principio, resolverlos es una parte esencial de la colaboración.

1. **Identificar un conflicto**

Cuando GIT detecta un conflicto, lo notificará al intentar hacer un merge o un pull. Por ejemplo:

CONFLICT (content): Merge conflict in archivo.txt

2. **Cómo resolver un conflicto**

1. Abrir el archivo que tiene el conflicto. GIT marcará las partes conflictivas de esta manera:

<<<<<<< HEAD

(Cambios en la rama actual)

=======

(Cambios en la rama que intentas fusionar)

>>>>>>> nombre-de-la-rama

2. Analizar los cambios y decidir cuál conservar. Puede ser uno, el otro o incluso una combinación de ambos.
3. Eliminar las marcas (<<<<<<<, =======, >>>>>>>) y guardar el archivo con los cambios solucionados.

3. Finalizar el proceso

Una vez resueltos todos los conflictos:

1. Añadir los archivos al área de preparación:
   git commit -m "Resolviendo conflictos en archivo.txt"

2. Completar el merge con un commit:
   git commit -m "Resolviendo conflictos en archivo.txt"

HERRAMIENTAS VISUALES

Aunque la línea de comandos de GIT es poderosa, existen herramientas visuales que simplifican el uso de GIT, especialmente para quienes están empezando o prefieren trabajar con interfaces gráficas. Estas herramientas ayudan a gestionar ramas, resolver conflictos y realizar commits sin necesidad de escribir comandos.

1. GitHub Desktop

Una aplicación oficial de GitHub para manejar repositorios desde el escritorio.

- **Características principales:**
  - Clonar repositorios fácilmente.
  - Visualizar el historial de commits.
  - Crear ramas y fusionarlas con unos pocos clics.
  - Resolver conflictos con una interfaz clara.

- **Ideal para usuarios de GitHub que buscan simplicidad.**

2. SourceTree

Una herramienta gratuita de Atlassian para gestionar repositorios GIT visualmente.

- **Características principales:**
  - Compatible con múltiples plataformas remotas (GitHub, GitLab, Bitbucket).
  - Gestión avanzada de ramas.
  - Visualización del flujo de trabajo mediante gráficos.

- **Ideal para proyectos más complejos con múltiples ramas y colaboradores.**

3. Visual Studio Code

Un editor de código que incluye integración con GIT.

- **Características principales:**
    - Ver el estado del repositorio directamente en el editor.
    - Resolver conflictos dentro del archivo mientras se edita el código.
    - Extensiones para integrar funcionalidades adicionales de GIT.

- **Ideal para desarrolladores que trabajan con código y GIT simultáneamente.**

4. GitKraken

Es una herramienta visual que combina un diseño intuitivo con funciones avanzadas de GIT.

- **Características principales:**
    - Soporte para repositorios locales y remotos.
    - Gestión visual de ramas y merges.
    - Opciones para resolver conflictos rápidamente.

- **Ideal para equipos grandes que requieren una herramienta poderosa y visual.**

Ventajas de usar herramientas visuales

- Menor margen de error al realizar operaciones.
- Aprendizaje más intuitivo para principiantes.
- Facilitan la colaboración y la resolución de conflictos.

BUENAS PRÁCTICAS

- Hacer commits pequeños y frecuentes.
- Escribir mensajes de commit descriptivos.
- Usar ramas para organizar el trabajo.
- No subir archivos innecesarios (configurar .gitignore).

REFERENCIAS

Acerca de GitHub y Git - Documentación de GitHub. (n.d.). GitHub Docs. https://docs.github.com/es/get-started/start-your-journey/about-github-and-git

Configuración de Git - Documentación de GitHub. (n.d.). GitHub Docs. https://docs.github.com/es/get-started/getting-started-with-git/set-up-git

Acerca de los repositorios remotos - Documentación de GitHub. (n.d.). GitHub Docs. https://docs.github.com/es/get-started/getting-started-with-git/about-remote-repositories