Introducción al control de versiones

Un sistema de control de versiones permite gestionar y registrar cada modificación realizada en los archivos. Su uso es fundamental en proyectos donde varias personas colaboran, ya que previene conflictos y pérdidas de información al mantener un historial claro de los cambios.

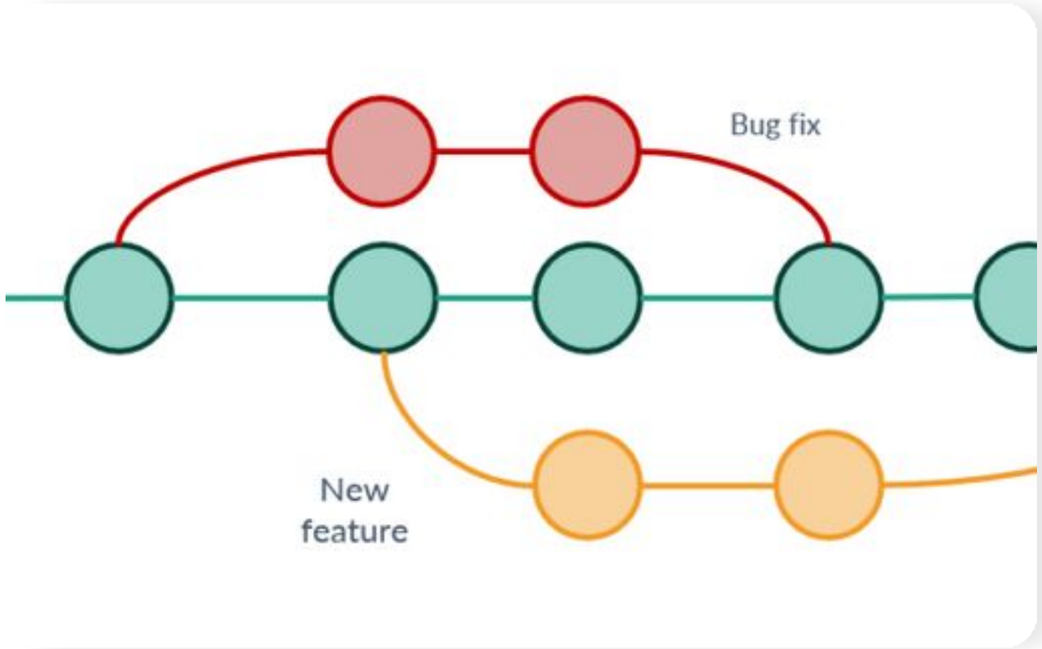
UTN

TUPaD - Programación 1

2025

¿Por qué usar Git y VCS?

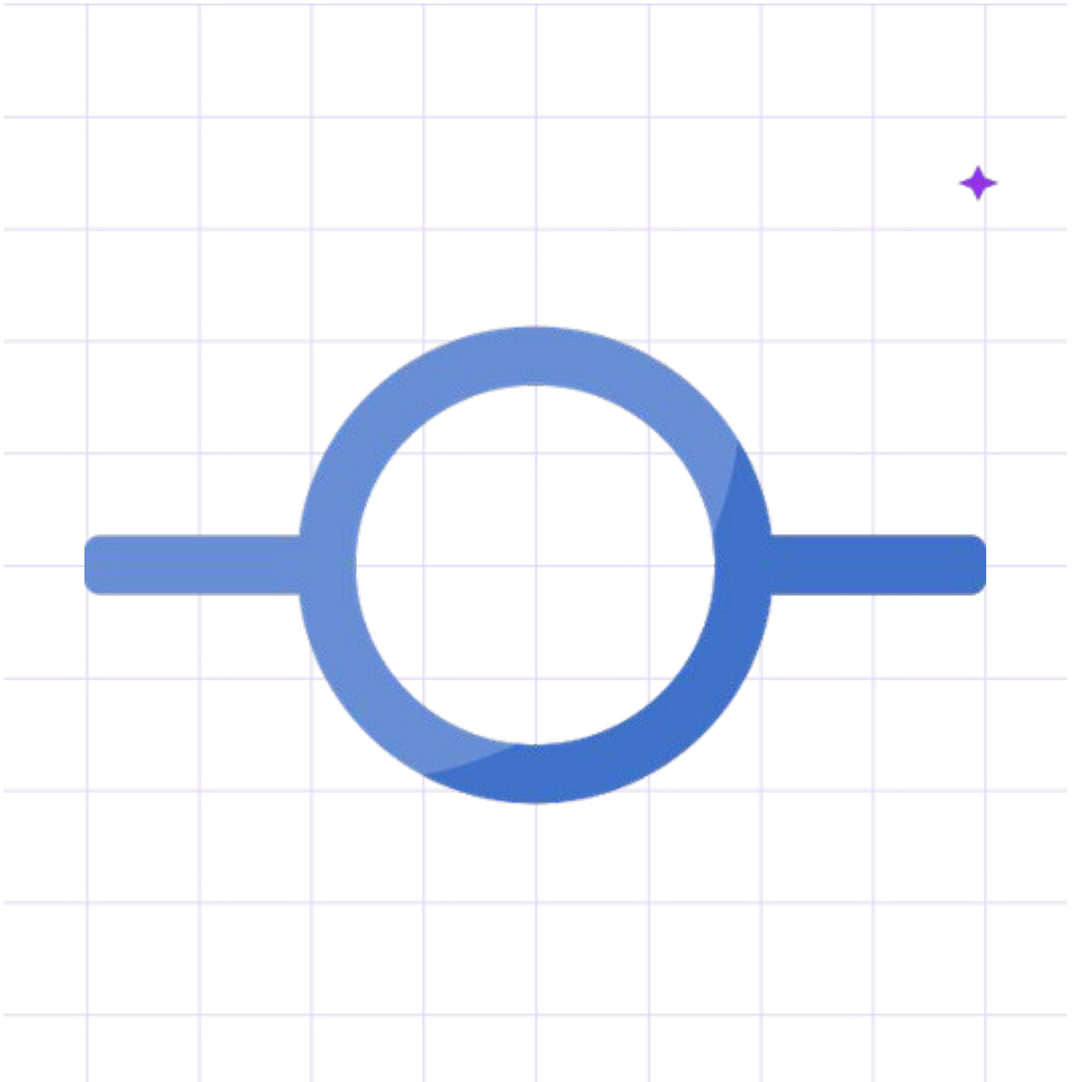
Git y los sistemas de control de versiones (VCS) facilitan el seguimiento de cambios, permitiendo identificar quién realizó cada modificación. Ayudan a recuperar versiones anteriores de un proyecto, lo que es fundamental para corregir errores o comparar avances. También organizan y coordinan la colaboración entre varios miembros del equipo, evitando que el trabajo de unos sobrescriba accidentalmente el de otros.



Git: Sistema de control distribuido

Git, desarrollado por Linus Torvalds, se ha convertido en el estándar de la industria para control de versiones. Su naturaleza distribuida permite mantener copias completas del repositorio en cada equipo. Las ramas livianas favorecen la experimentación y el trabajo simultáneo, mientras que la posibilidad de trabajar sin conexión habilita mayor flexibilidad y eficiencia. Además, Git facilita la colaboración entre múltiples desarrolladores, integrando fácilmente los cambios.

Commits: Concepto y buenas prácticas



Los commits permiten capturar el estado de un proyecto en un momento específico, facilitando el seguimiento de cambios y la colaboración. Realizar commits frecuentes y con mensajes claros ayuda a mantener la organización y simplifica la identificación de errores en el código.

Frecuencia

Hacer commits de forma regular evita perder avances y permite rastrear modificaciones de manera eficiente.

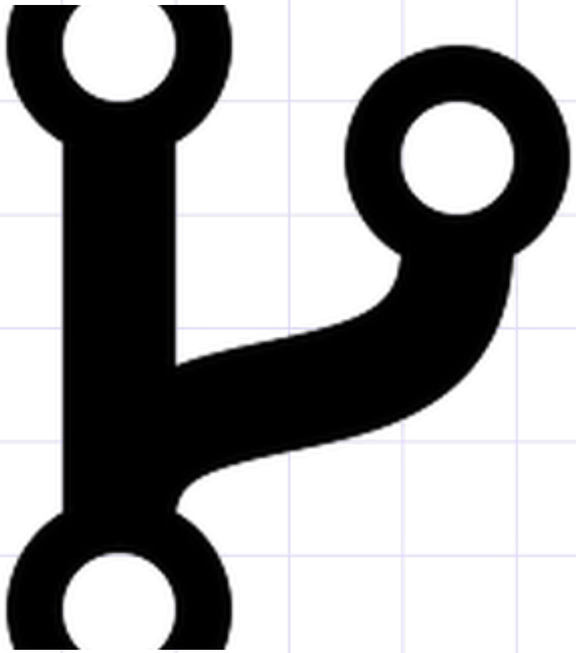
Mensajes descriptivos

Utilizar mensajes claros en cada commit ayuda a comprender las razones detrás de los cambios realizados.

Trazabilidad

Los commits organizados facilitan la detección, revisión y corrección de errores en el proyecto.

Ramas y manejo de conflicto



El uso de ramas facilita la implementación de nuevas características sin afectar la línea principal de desarrollo. Al unir los cambios realizados en distintas ramas, pueden surgir conflictos que requieren intervención y Git proporciona herramientas para identificarlos y solucionarlos de manera eficiente.

Desarrollo paralelo

Las ramas posibilitan que varios colaboradores trabajen en distintas tareas simultáneamente.

Fusión de cambios

Al integrar ramas, los cambios independientes se combinan en un solo proyecto.

Resolución de conflictos

Git proporciona métodos interactivos para detectar y solucionar conflictos durante la fusión.

Repositorios remotos y GitHub

Las plataformas como GitHub permiten almacenar y gestionar proyectos de código de manera remota, facilitando la colaboración y sincronización entre varios usuarios. Utilizar comandos básicos posibilita compartir avances y mantener los archivos actualizados en el equipo de trabajo.

Almacenamiento remoto seguro

Los proyectos se guardan en la nube, protegidos y accesibles cuando los necesites.

Colaboración en equipo

Varios usuarios pueden contribuir al mismo proyecto simultáneamente, facilitando la integración de ideas.

Comandos básicos (clone, pull, push)

Permiten descargar repositorios, obtener cambios recientes y subir modificaciones al repositorio remoto.

Forks y Pull Requests

Forks permiten a los usuarios crear una copia de un repositorio para trabajar en cambios de manera independiente. Los Pull Requests facilitan la colaboración, al permitir proponer, revisar y combinar cambios sugeridos antes de integrarlos al proyecto original.

Forks

Permiten experimentar y desarrollar nuevas funciones sin afectar el proyecto principal.

Pull Requests

Brindan un flujo estructurado para discutir, revisar y aprobar cambios antes de integrarlos.

Colaboración Segura

Fomentan la colaboración abierta y controlada entre desarrolladores independientes.

Ventajas del modelo distribuido

Trabajo sin conexión

Posibilidad de avanzar incluso sin acceso continuo a la red.

Redundancia y fiabilidad

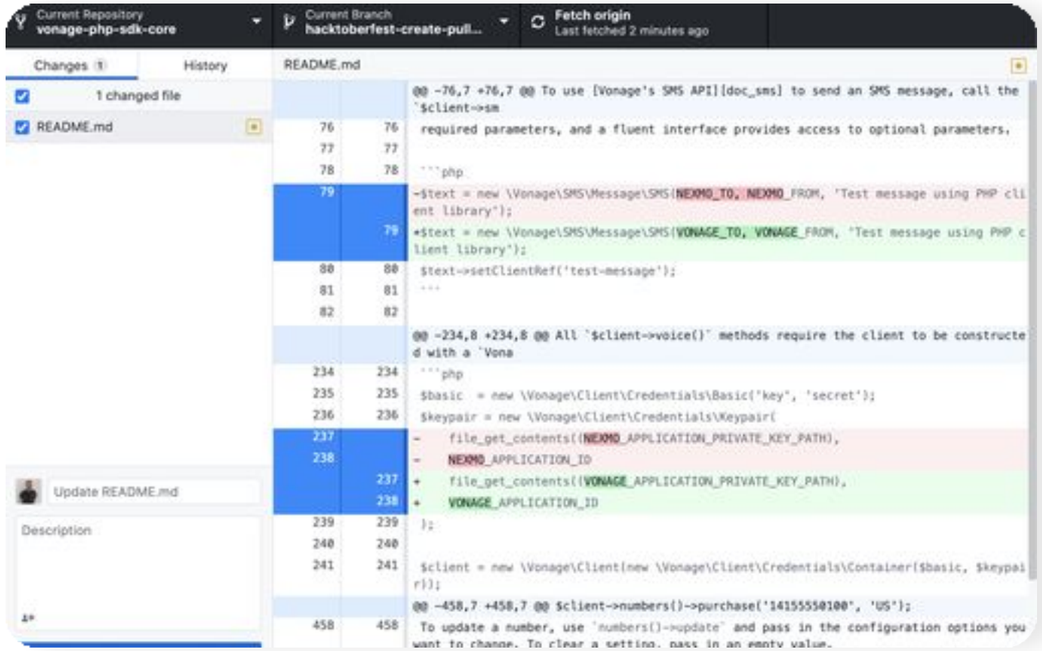
Incrementa la tolerancia a fallos y minimiza el riesgo de pérdidas de datos.

Colaboración descentralizada

Equipos pueden contribuir simultáneamente desde diferentes ubicaciones.

Desarrollo paralelo eficiente

Facilita el avance simultáneo en diferentes áreas del proyecto.



¿Qué es GitHub Desktop?

GitHub Desktop es una aplicación gráfica diseñada para facilitar el manejo de proyectos en Git y GitHub. Su interfaz intuitiva permite a los usuarios realizar tareas habituales sin necesidad de utilizar la línea de comandos, resultando especialmente útil para quienes buscan una experiencia visual y sencilla al gestionar repositorios y colaboraciones.

Creación de cuenta y repositorio en GitHub

01 Registro de cuenta en GitHub

Accede al sitio web de GitHub y completa el formulario de registro proporcionando datos básicos como correo electrónico y contraseña.

02 Creación de un repositorio remoto

Una vez registrado, crea un nuevo repositorio eligiendo un nombre, añadiendo una breve descripción y estableciendo la visibilidad como pública o privada.

03 Configuración inicial para GitHub Desktop

Con la cuenta y el repositorio listos, podrás vincularlos fácilmente a GitHub Desktop para gestionar proyectos de manera eficiente desde tu ordenador.

Instalación y configuración de GitHub Desktop

01 Descarga la aplicación

Accede al sitio oficial de GitHub Desktop y selecciona la versión compatible con tu sistema operativo para iniciar la descarga.

02 Instalación sencilla

Ejecuta el archivo descargado y sigue los pasos indicados en pantalla para completar el proceso de instalación sin complicaciones.

03 Inicio de sesión en tu cuenta

Abre GitHub Desktop e ingresa tus credenciales de GitHub para vincular tu cuenta y empezar a gestionar tus proyectos fácilmente.

Clonar repositorio y comenzar a trabajar

01 Abrir GitHub Desktop

Inicia la aplicación GitHub Desktop para gestionar tus repositorios desde una interfaz gráfica intuitiva.

02 Seleccionar 'Clonar repositorio'

Elige la opción para clonar un repositorio remoto y cópialo a tu equipo local de manera rápida y segura.

03 Modificar archivos localmente

Realiza cambios, edita archivos y prueba nuevas funcionalidades en tu entorno antes de compartirlos en GitHub.

Flujo básico: edición, commit y push

01 Edita tus archivos

Realiza modificaciones en tus archivos utilizando el editor de tu elección para mejorar o corregir el proyecto.

02 Registra cambios con commit

Guarda localmente tus cambios realizados mediante la opción de commit en GitHub Desktop, añadiendo un mensaje descriptivo.

03 Sincroniza con push

Envía los cambios de tu repositorio local al repositorio remoto usando la función push, asegurando que la versión en la nube esté actualizada.

Trabajo colaborativo: ramas individuales

El trabajo en ramas individuales permite que cada desarrollador organice su avance de forma independiente, mejore la trazabilidad de los cambios y simplifique el proceso de integración de nuevas funciones al proyecto principal.

Creación de ramas propias

Cada integrante inicia una nueva rama específica para su tarea asignada.

Registro de cambios independiente

Los cambios se documentan en la rama del desarrollador sin afectar el código principal.

Publicación y revisión

Las ramas se publican y están listas para revisión y colaboración si es necesario.

Resumen y trazabilidad del historial

El historial de Git revela cada cambio realizado, identificando el momento y el autor, lo que facilita la auditoría del proceso y la reconstrucción precisa de los pasos del desarrollo.