**Clase 4:** Introducción a Git

Objetivos de la clase

* Comprender qué es Git y por qué es importante en el desarrollo de software.
* Familiarizarse con los conceptos básicos de control de versiones y cómo Git los implementa.
* Aprender los comandos básicos de Git para la gestión de repositorios locales.

Contenido de la clase

* *¿Qué es Git?*
  + Definición de Git y su importancia en el control de versiones. Comparación con otros sistemas de control de versiones. Breve historia y contexto de Git. Conceptos básicos de control de versiones
* *¿Qué es el control de versiones y por qué es necesario?*
  + Versionado de archivos: seguimiento de cambios a lo largo del tiempo. Ramificación (branching) y fusión (merging) en el control de versiones.
* *Instalación y configuración de Git*
  + Descarga e instalación de Git en diferentes sistemas operativos. Configuración inicial de Git: nombre de usuario, dirección de correo electrónico, etc. Configuración de herramientas adicionales, como un editor de texto por defecto.
* *Conceptos básicos de Git*
  + Repositorios: local vs. remoto. Commits: capturar cambios en el repositorio. Ramas (branches): trabajando en diferentes líneas de desarrollo. Fusiones (merges): combinando cambios de diferentes ramas.
* *Comandos básicos de Git*
  + git init: inicializar un repositorio Git. git clone: clonar un repositorio existente. git add: añadir cambios al área de preparación (staging area). git commit: confirmar cambios en el repositorio. git status: ver el estado actual del repositorio. git log: ver el historial de commits. git branch: trabajar con ramas. git merge: fusionar ramas. git push: enviar cambios al repositorio remoto. git pull: obtener cambios del repositorio remoto.

**Introducción**

Bienvenidos y bienvenidas a una nueva clase, en la cual daremos nuestros primeros pasos en Git, la cual es una herramienta fundamental en el repertorio de herramientas de cualquier desarrollador en la actualidad. En esta clase, exploraremos los fundamentos de Git y cómo puede revolucionar la forma en que gestionamos y colaboramos en proyectos de desarrollo de software.

Comenzaremos planteándonos algunas preguntas clave: *¿Qué es Git y por qué es tan importante en el desarrollo de software? ¿Cómo difiere de otros sistemas de control de versiones? ¿Cuáles son sus beneficios en términos de colaboración, seguimiento de cambios y gestión de código?*

A lo largo de este encuentro, aprenderemos no solo a utilizar los comandos básicos de Git para gestionar versiones de nuestro código, sino también a entender conceptos clave como repositorios, ramas, commits y merges. A su vez, exploraremos cómo Git facilita el trabajo en equipo, permitiendo a múltiples desarrolladores trabajar simultáneamente en un mismo proyecto de manera eficiente y organizada.

Este recorrido hacia el dominio de Git no solo mejorará nuestra capacidad para manejar proyectos de manera efectiva, sino que también nos equiparará con habilidades fundamentales para el desarrollo profesional

**¿Qué es Git y por qué es importante?**

Git es un sistema de control de versiones, lo que significa que ayuda a rastrear los cambios en los archivos a lo largo del tiempo. Imaginemos que estamos trabajando en un proyecto de software con un equipo de personas. Git te permite colaborar de manera efectiva, manteniendo un registro de quién hizo qué cambios y cuándo los hizo. Esto es fundamental para trabajar de manera organizada y evitar conflictos.

**Comparación con otros sistemas de control de versiones**

Antes de Git, había otros sistemas de control de versiones, pero ninguno era tan poderoso y versátil. Git se destaca por su velocidad, su capacidad para manejar proyectos grandes y su sistema de ramificación flexible, que permite a los equipos trabajar en diferentes características de manera independiente.

**Breve historia y contexto de Git**

Git fue creado por Linus Torvalds, la misma persona que desarrolló el kernel de Linux. Lo creó porque necesitaba una herramienta de control de versiones para ayudar en el desarrollo de Linux. Desde entonces, Git se ha convertido en la herramienta estándar de facto para el control de versiones en la industria del software.

**Conceptos básicos de control de versiones**

Antes de sumergirnos en Git, es importante comprender algunos conceptos básicos de control de versiones, como el *versionado de archivos* y la *ramificación* y *fusión* de cambios. Estos conceptos forman la base de cómo funciona Git y por qué es tan útil para el desarrollo de software.

**¿Qué es el control de versiones y por qué es necesario?**

A continuación, profundizaremos en el concepto de control de versiones y entenderemos por qué es una práctica fundamental en el desarrollo de software.

*Versionado de archivos: seguimiento de cambios a lo largo del tiempo*

Imaginemos que estás trabajando en un proyecto de software y realizas algunos cambios en un archivo. Sin un sistema de control de versiones, sería difícil rastrear esos cambios o volver a versiones anteriores si algo sale mal, por ejemplo, rompiendo el sistema o generando un bug. El control de versiones nos permite mantener un historial de todos los cambios realizados en los archivos, lo que facilita la gestión y la colaboración en distintos proyectos.

*Ramificación (branching) y fusión (merging) en el control de versiones*

A medida que el proyecto crece, es probable que necesitemos trabajar en diferentes características o arreglar diferentes problemas al mismo tiempo. La ramificación nos permite crear copias separadas de tu código, llamadas *"ramas"*, para trabajar en estas características o problemas de forma aislada. Luego, la fusión te permite combinar esos cambios de vuelta en la rama principal una vez que estén listos. Esto facilita la colaboración entre equipos y la gestión de cambios complejos en el código.

**Instalación y configuración de Git**

*¡Es hora de poner manos a la obra y empezar a trabajar con Git en tu computadora!*

*Descarga e instalación de Git en diferentes sistemas operativos*

Git está disponible para una amplia gama de sistemas operativos, incluyendo *Windows, macOS y Linux.* Podemos descargar la última versión de Git desde el sitio web oficial (Enlace: <https://git-scm.com/> ).

Una vez descargado el instalador, simplemente sigue las instrucciones para completar la instalación en nuestro sistema operativo.

*Configuración inicial de Git: nombre de usuario, dirección de correo electrónico, etc.*

Después de instalar Git, es importante configurar algunos ajustes iniciales, como nuestro nombre de usuario y dirección de correo electrónico. Esto permite que Git identifique quién realizó cada cambio en el repositorio. Puedes configurar estos ajustes utilizando los siguientes comandos:

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

git config --global user.email "tu@email.com"

*Configuración de herramientas adicionales, como un editor de texto por defecto*

Además de configurar tu nombre de usuario y dirección de correo electrónico, es posible que también desees configurar un editor de texto por defecto para utilizar con Git. Puedes hacerlo utilizando el siguiente comando:

git config --global core.editor "nombre\_del\_editor"

Por ejemplo, si queremos utilizar el editor Vim, podemos ejecutar:

git config --global core.editor "vim"

**Conceptos básicos de Git**

¡Estamos yendo a paso a paso en este recorrido por Git! En este nuevo apartado, nos sumergiremos en los conceptos básicos de Git y entenderemos cómo funciona este poderoso sistema de control de versiones.

*Repositorios: local vs. remoto*

En Git, un repositorio es donde se almacena todo el historial de cambios de tu proyecto. Hay dos tipos principales de repositorios: *locales y remotos*. Un repositorio local reside en nuestra computadora y es donde trabajas en tu proyecto. Por otro lado, un repositorio remoto es una copia del repositorio en un servidor, como GitHub o GitLab, que nos permite colaborar con otros desarrolladores y mantener una copia de seguridad de nuestro trabajo.

*Commits: capturar cambios en el repositorio*

Un commit en Git es un registro de un conjunto de cambios en nuestro repositorio. Cada commit tiene un mensaje asociado que describe los cambios realizados. Los commits son como hitos en la historia de nuestro proyecto, y permiten rastrear quién hizo qué cambios y cuándo los hicieron.

*Ramas (branches): trabajando en diferentes líneas de desarrollo*

Las ramas en Git son copias separadas de nuestro código que permiten trabajar en nuevas características o arreglar errores sin afectar la rama principal del proyecto. Esto es útil porque nos permite experimentar y probar cosas nuevas sin arriesgar el código principal del proyecto.

*Fusiones (merges): combinando cambios de diferentes ramas*

Cuando terminamos de trabajar en una nueva característica o arreglo de errores en una rama, podemos fusionar esos cambios de vuelta a la rama principal del proyecto. Esto se llama una fusión (merge) en Git, y es una forma de combinar diferentes líneas de desarrollo en una sola.

**Comandos básicos de Git**

En esta sección, aprenderemos los comandos básicos de Git que nos permitirán gestionar nuestro repositorio local y colaborar con otros desarrolladores de manera efectiva.

*git init: inicializar un repositorio Git*

El comando **git init** se utiliza para iniciar un nuevo repositorio Git en un directorio. Simplemente ejecuta este comando en la carpeta de tu proyecto y Git creará un nuevo repositorio local donde podrás comenzar a trabajar en tu proyecto.

*git clone: clonar un repositorio existente*

Si queremos trabajar en un proyecto que ya existe en un repositorio remoto, podemos usar el comando **git clone** para crear una copia local del repositorio en nuestra computadora. Solo necesitamos proporcionar la URL del repositorio remoto y Git se encargará del resto.

*git add: añadir cambios al área de preparación (staging area)*

Antes de confirmar nuestros cambios en el repositorio, debemos agregarlos al área de preparación (staging area) utilizando el comando **git add**. Esto permite seleccionar qué cambios deseamos incluir en nuestros próximo commit.

*git commit: confirmar cambios en el repositorio*

Una vez que hemos agregado nuestros cambios al área de preparación, podemos confirmarlos en el repositorio utilizando el comando **git commit**. Es importante incluir un mensaje descriptivo que explique los cambios que hicimos en este commit.

*git status: ver el estado actual del repositorio*

El comando **git status** muestra el estado actual de nuestro repositorio, incluidos los cambios que aún no se han confirmado, las ramas en las que estamos trabajando y más. Es una herramienta útil para mantenerse al tanto de lo que está sucediendo en el proyecto.

*git log: ver el historial de commits*

El comando **git log** muestra el historial de commits en el repositorio, incluidos los mensajes de commit, los autores y las fechas de los cambios. Esto te permite ver quién hizo qué cambios en el proyecto a lo largo del tiempo.

*git branch: trabajar con ramas*

El comando **git branch** permite crear, listar y eliminar ramas en tu repositorio. Las ramas son útiles para trabajar en diferentes características o arreglos de errores de forma aislada.

*git merge: fusionar ramas*

Cuando hemos terminado de trabajar en una rama y estamos listos para combinar los cambios en otra rama, podemos utilizar el comando **git merge** para fusionar las ramas. Esto permite combinar diferentes líneas de desarrollo en una sola.

*git push: enviar cambios al repositorio remoto*

Una vez que confirmamos nuestros cambios en el repositorio local, podemos enviarlos al repositorio remoto utilizando el comando **git push**. Esto actualiza el repositorio remoto con los cambios y los hace disponibles para otros desarrolladores.

*git pull: obtener cambios del repositorio remoto*

Si otros desarrolladores han realizado cambios en el repositorio remoto, podemos obtener estos cambios en nuestro repositorio local utilizando el comando **git pull**. Esto nos permite mantener el repositorio local actualizado con los últimos cambios en el repositorio remoto.