Lección 4: Control de versiones con Git y Github.

El control de versiones es imprescindible para la gestión ordenada y colaborativa del código fuente. Git, como sistema de control de versiones distribuido, destaca por su capacidad para seguir los cambios, facilitar la colaboración en equipos y mantener un historial detallado del desarrollo del proyecto. GitHub, es un repositorio remoto, que complementa a Git al proporcionar un espacio centralizado para el almacenamiento y la gestión de proyectos. En esta lección, explorarán las funcionalidades y aplicaciones prácticas de Git y GitHub, proporcionando a los participantes las habilidades necesarias para optimizar su flujo de trabajo y colaborar de manera eficiente en proyectos de desarrollo de software y de ciencia de datos.

¿Qué es sistema de control de versiones?

Es una herramienta que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que sea posible revisar y recuperar versiones específicas del código fuente de un proyecto. El control de versiones se puede extender a cualquier archivo, no solamente código fuente. Sin embargo, es muy común en la gestión de software.

Con un sistema de control de versiones es posible recuperar archivos sobreescritos, recuperar cambios cuando se ha guardado un documento o código, revisar versiones, comparar archivos a lo largo del tiempo e incluso, cuando muchas personas colaboran en un proyecto, es posible saber quién hizo qué cambios y cuando un cambio se convirtió en un problema o en un error. Los controles de versiones también permiten recuperar archivos perdidos o eliminados.

Todas estas preguntas permiten determinar el contenido de los datos de forma general y orientar las decisiones para el análisis de los datos.

Con las bases establecidas, el siguiente paso es la visualización. Se pueden emplear gráficos y diagramas para representar visualmente la información. Un histograma puede revelar la distribución de datos numéricos, mientras que un diagrama de dispersión puede mostrar relaciones entre variables. Estas representaciones gráficas son claves para identificar patrones de manera intuitiva.

Sistemas de Control de Versiones Locales.

Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son cuidadosos). Este método es muy común porque es muy sencillo, pero también es propenso a errores. Ya que es fácil que cualquiera altere los archivos sin saber quién fue, o se sobreescriban archivos sin poder regresar a una versión anterior.

El control de versiones también ayuda a eliminar malas prácticas como el almacenamiento local y el versionamiento escueto, por ejemplo, archivos con nombres como: final_este_si, ignorar_los_otros, versión_definitiva, versión_de_entrega; que son un problema si se quiere saber cuál es realmente la última versión o cual de ellas está completa y revisada.

Ventajas del control de versiones.

El control de versiones beneficia muchos aspectos de la producción de software, por ejemplo sirve para:

Crear flujos de trabajo evitando el caos a todos los usuarios que usan su propio proceso de desarrollo con herramientas diferentes e incompatibles. Los sistemas de control de versiones proporcionan permisos y cumplimiento de procesos, por lo que todos permanecen en sintonía.

Trabajo con versiones: cada versión tiene una descripción de lo que hacen los cambios en la versión, cómo corregir un error o agregar una característica. Estas descripciones ayudan al equipo a seguir los cambios del código por versión en lugar de por cambios de archivo individuales. El código almacenado en versiones se puede ver y restaurar desde el control de versiones en cualquier momento y según sea necesario. Las versiones facilitan basar el nuevo trabajo en cualquier versión del código.

Trabajar juntos: el control de versiones sincroniza las versiones y garantiza que los cambios no entren en conflicto con los cambios de otros usuarios. El equipo se basa en el control de versiones para ayudar a resolver y evitar conflictos, incluso cuando los usuarios realizan cambios al mismo tiempo.

Mantener un historial: el control de versiones mantiene un historial de los cambios a medida que el equipo guarda nuevas versiones del código. Los miembros del equipo pueden revisar el historial para averiguar la persona que realizó los cambios, por qué los hizo y en qué momento. El historial ofrece a los equipos la confianza de experimentar, ya que es fácil revertir a una versión anterior correcta en cualquier momento. El historial permite que cualquier persona base el trabajo en cualquier versión del código, cómo corregir un error en una versión anterior.

Automatización de tareas.

Las características de automatización del control de versiones ahorran tiempo y generan resultados coherentes. La automatización de pruebas, el análisis de códigos y la implementación cuando se guardan nuevas versiones en el control de versiones son solo tres ejemplos.

Sofware Git.

Git se ha convertido en el estándar mundial para el control de versiones. Es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor.

Aspectos básicos de Git.

Cada vez que se guarda el trabajo, Git crea una confirmación o commit. Una confirmación es una instantánea de todos los archivos en un momento dado. Si un archivo no ha cambiado de una confirmación a la siguiente, Git usa el archivo almacenado anteriormente. Las confirmaciones crean vínculos a otras confirmaciones, formando un gráfico del historial de desarrollo. Es posible revertir el código a una confirmación anterior, inspeccionar cómo cambian los archivos de una confirmación a la siguiente y revisar información cómo, dónde y cuándo se realizaron los cambios. Las confirmaciones se identifican en Git mediante un hash criptográfico único del contenido de la confirmación. Dado que todo tiene hash, es imposible realizar cambios, perder la información o dañar los archivos sin que Git lo detecte.

Ramas.

Cada desarrollador guarda los cambios en su propio repositorio de código local. Como resultado, puede haber muchos cambios diferentes basados en la misma confirmación. Git proporciona herramientas para aislar los cambios y volver a combinarlos posteriormente. Las ramas, que son punteros ligeros para el trabajo en curso, administran esta separación. Una vez finalizado el trabajo creado en una rama, se puede combinar de nuevo en la rama principal (o troncal) del equipo.

Archivos y confirmaciones

Los archivos de Git se encuentran en uno de estos tres estados: modificados, almacenados provisionalmente o confirmados. Cuando se modifica un archivo por primera vez, los cambios solo existen en el directorio de trabajo. Todavía no forman parte de una confirmación ni del historial de desarrollo. El desarrollador debe almacenar provisionalmente los archivos modificados que se incluirán en la confirmación. El área de almacenamiento provisional contiene todos los cambios que se incluirán en la siguiente confirmación. Una vez que el desarrollador esté satisfecho con los archivos almacenados provisionalmente, los archivos se empaquetan como una confirmación con un mensaje que describe lo que ha cambiado. Esta confirmación pasa a formar parte del historial de desarrollo.

El almacenamiento provisional permite a los desarrolladores elegir qué cambios de archivo se guardarán en una confirmación para desglosar los cambios grandes en una serie de confirmaciones más pequeñas. Al reducir el ámbito de las confirmaciones, es más fácil revisar el historial de confirmaciones para buscar cambios de archivo específicos.