CONTROL DE VERSIONES

Entorno de desarrollo tema 5

Tabla de contenido

1 Git

El control de las diferentes versiones de un software se realiza utilizando una serie de herramientas como por ejemplo GIT.

Git es un sistema de control de versiones distribuido y de código abierto. A continuación, se detallan algunas de sus características principales:

1. **Distribuido:** cada usuario tiene una copia completa del repositorio, lo que permite trabajar de forma autónoma y sin conexión a internet.
2. **Ramificación y fusión**: permite crear ramas de desarrollo independientes y luego fusionarlas de manera sencilla y eficiente.
3. **Historial de cambios**: registra el historial completo de cambios realizados en el código, lo que permite rastrear los cambios y revertirlos si es necesario.
4. **Colaboración:** permite la colaboración entre varios desarrolladores en un mismo proyecto, facilitando la gestión de conflictos y la integración de cambios.
5. **Integración con herramientas de terceros**: puede integrarse fácilmente con herramientas como GitHub, Bitbucket, JIRA, Travis CI, etc.
6. **Seguridad:** proporciona mecanismos para proteger y garantizar la integridad del código fuente, incluyendo la autenticación y el cifrado.
7. **Eficiencia:** utiliza algoritmos eficientes para almacenar y recuperar cambios, lo que permite trabajar con proyectos grandes y complejos de forma rápida y eficiente.
8. **Flexibilidad:** puede ser utilizado en una amplia variedad de proyectos y tecnologías, desde pequeños proyectos personales hasta grandes proyectos empresariales.

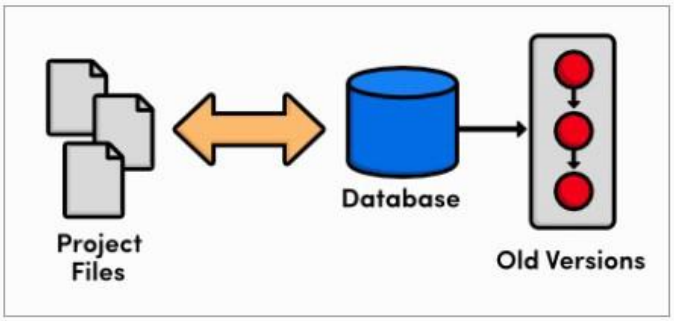
2 Sistemas de control

Existen diferentes sistemas de control que podemos emplear:

- Versiones locales, centralizadas, distribuidas.
-

2.1 Sistemas de Control de Versiones Locales

Los sistemas de control de versiones locales en vez de mantener las versiones como archivos independientes, los almacenaban en una base de datos. Cuando era necesario revisar una versión anterior del proyecto se usaba el sistema de control de versiones en vez de acceder directamente al archivo, de esta manera en cualquier momento solo se tenia una copia del proyecto, eliminando la posibilidad de confundir o eliminar versiones.

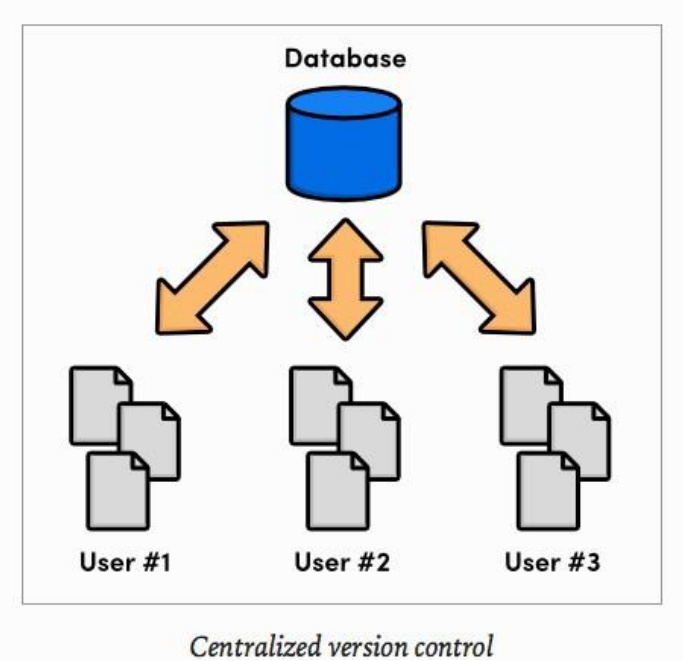


Local version control

En este punto el control de versiones se llevaba a cabo en el ordenador de cada uno de los desarrolladores y no existía una manera eficiente de compartir el código entre ellos.

2.2 Sistemas de Control de Versiones Centralizados

Para facilitar la colaboración de múltiples desarrolladores en un solo proyecto los sistemas de control de versiones evolucionaron: en vez de almacenar los cambios y versiones en el disco duro de los desarrolladores, estos se almacenaban en un servidor.



Centralized version control

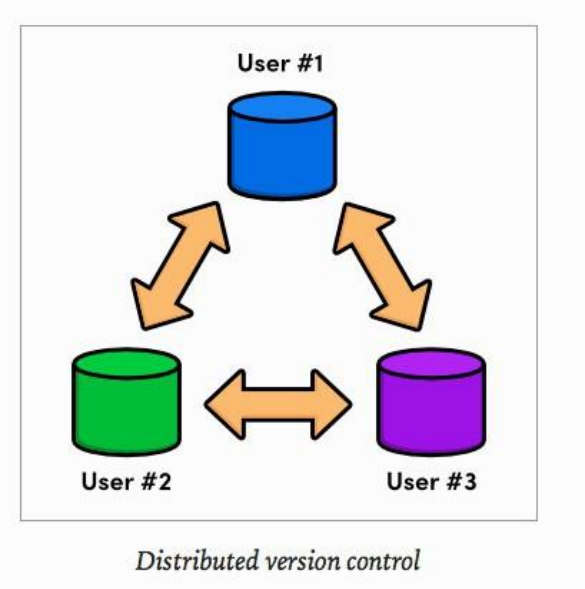
Sin embargo, aunque el avance frente a los sistemas de control de versiones locales fue enorme, los sistemas centralizados trajeron consigo nuevos retos: ¿Cómo trabajaban múltiples usuarios en un mismo archivo al mismo tiempo?

Los sistemas de control de versiones centralizados abordaron este problema impidiendo que los usuarios invalidaran el trabajo de los demás. Si dos personas editaban el mismo archivo y se presentaba un conflicto alguien debía solucionar este problema de manera manual y el desarrollo no podía continuar hasta que todos los conflictos fueran resueltos y puestos a disposición del resto del equipo.

Esta solución funcionó en proyectos que tenían relativamente pocas actualizaciones y por ende pocos conflictos, pero resulto muy engorroso para proyectos con docenas de contribuyentes activos que realizaban actualizaciones a diario.

2.3 Sistemas de Control de Versiones Distribuidos

La siguiente generación de sistemas de control de versiones se alejó de la idea de un solo repositorio centralizado y optó por darle a cada desarrollador una copia local de todo el proyecto, de esta manera se construyó una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada, pero teniendo un mecanismo de resolución de conflictos mucho más elegante que una su versión anterior.



Distributed version control

Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclar los conflictos que se presenten solo cuando se requiera. Cómo cada usuario tiene una copia completa del proyecto el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de perdida de datos es mucho menor que en cualquiera de sus predecesores.

3 Tipos de colaboración en un SCV

Hay varios tipos de colaboración que pueden llevarse a cabo en un sistema de control de versiones (SCV). A continuación, se detallan algunos de ellos:

3.1 Flujo de trabajo centralizado o Wordkflow centralizado

¿Qué es?

Flujo de trabajo centralizado o Wordkflow centralado El flujo de trabajo centralizado es un modelo de colaboración en el que todos los desarrolladores trabajan en una única rama principal del repositorio. En este modelo, los cambios son enviados a través de un sistema de control

centralizado y sólo uno o unos pocos desarrolladores tienen permiso para fusionar los cambios a la rama principal

Pasos

1. El desarrollador descarga una copia de la rama principal del repositorio.
2. Trabaja en su propia copia local y realiza los cambios necesarios.
3. Cuando ha terminado, envía los cambios al servidor centralizado para que sean revisados y aprobados.
4. Uno o varios miembros del equipo con permiso de fusión revisan y aprueban los cambios.
5. Si los cambios son aprobados, se fusionan en la rama principal del repositorio. Si hay conflictos, se resuelven antes de la fusión.

Ventajas	Desventajas
• Este modelo de colaboración puede ser efectivo en equipos pequeños o proyectos simples,	• Puede tener limitaciones en proyectos más grandes o complejos donde se necesitan múltiples ramas de desarrollo para trabajar de manera efectiva. • Además, este modelo puede ser menos eficiente para trabajar en paralelo o de forma simultánea en diferentes funcionalidades o características del proyecto.

3.2 Flujo de trabajo con gestor de integración

¿Qué es?

El flujo de trabajo con un gestor de integración (también conocido como gestor de integración continua) es un modelo de colaboración que se enfoca en la integración y verificación continua de los cambios realizados en el repositorio. Este flujo de trabajo se utiliza comúnmente en proyectos grandes o complejos donde se necesitan múltiples ramas de desarrollo y se requiere una integración continua y frecuente del código para asegurar que no haya errores en el producto final.

Pasos

1. El desarrollador descarga una copia de la rama principal del repositorio y crea su propia rama de trabajo.
2. Trabaja en su propia rama de trabajo y realiza los cambios necesarios.

3. Cuando ha terminado, envía los cambios a través del gestor de integración.
4. El gestor de integración recibe los cambios, los integra en la rama principal del repositorio y ejecuta una serie de pruebas automatizadas para verificar que los cambios no hayan afectado negativamente a la calidad del software.
5. Si las pruebas son exitosas, el gestor de integración notifica al equipo de que los cambios han sido integrados y están listos para su revisión.
6. Otro miembro del equipo revisa los cambios y si es necesario, se hacen correcciones y se vuelve a enviar a través del gestor de integración para otra ronda de integración y pruebas.

Ventajas	Desventajas
• Este flujo de trabajo con un gestor de integración permite a los desarrolladores trabajar de manera más independiente, ya que pueden trabajar en sus propias ramas de trabajo y enviar cambios sin tener que esperar a que otro miembro del equipo apruebe la fusión. • Además, al tener pruebas automatizadas en el proceso de integración, se puede asegurar que los cambios realizados no rompan la funcionalidad existente y se mantenga una alta calidad del software.	• **Dependencia del gestor de integración:** En este flujo de trabajo, el gestor de integración se convierte en un punto crítico para la integración y la entrega del código. Si el gestor de integración no está disponible o tiene problemas, puede afectar el flujo de trabajo y retrasar la entrega del código. • **Complejidad:** Este flujo de trabajo puede ser más complejo que otros modelos de colaboración, ya que involucra múltiples ramas de trabajo, pruebas automatizadas y a veces una configuración más avanzada del gestor de integración. • **Configuración y mantenimiento del gestor de integración:** Para utilizar este flujo de trabajo, es necesario configurar y mantener el gestor de integración, lo que puede requerir tiempo y recursos adicionales. • **Dificultad para solucionar problemas**: Si ocurre algún problema en el proceso de integración o las pruebas automatizadas, puede ser más difícil de identificar y solucionar que en otros modelos de colaboración.

3.3 Flujo de trabajo con dictador y tenientes

¿Qué es?

El flujo de trabajo con dictador y tenientes (también conocido como "modelo de tenientes") es un modelo de colaboración que se utiliza en proyectos de código abierto y software libre. Este modelo se basa en un líder o "dictador" que tiene la autoridad final en la toma de decisiones, pero delega

responsabilidades a un grupo de "tenientes" que son responsables de diferentes partes del proyecto.

Pasos

1. El dictador es el encargado de la rama principal del repositorio y es el único con permiso para fusionar cambios en la rama principal.
2. Los tenientes son responsables de diferentes áreas del proyecto y tienen permiso para fusionar cambios en sus propias ramas de trabajo.
3. Los desarrolladores envían los cambios a través de las ramas de trabajo de los tenientes, y los tenientes revisan y aprueban los cambios antes de fusionarlos en sus propias ramas.
4. Los tenientes envían los cambios a la rama principal del repositorio a través del dictador, quien revisa y aprueba los cambios antes de fusionarlos en la rama principal.

Ventajas	Desventajas
• Este flujo de trabajo permite que el dictador tenga el control final y tome las decisiones importantes, mientras que los tenientes tienen un nivel de autonomía y responsabilidad para tomar decisiones en sus propias áreas de trabajo. • Además, este modelo también permite que los desarrolladores tengan una mayor libertad para trabajar en sus propias ramas de trabajo y enviar cambios sin tener que esperar a que otros miembros del equipo aprueben la fusión.	• Dependencia del dictador para tomar decisiones importantes y la posibilidad de que los tenientes tomen decisiones que puedan afectar a todo el proyecto sin una revisión adecuada del dictador. • Además, este modelo puede ser menos eficiente para trabajar en proyectos grandes o complejos donde se necesitan múltiples ramas de desarrollo y una integración continua y frecuente del código.

Otra forma de clasificarlos sería

Colaboración en equipo:

Varias personas trabajan en un mismo proyecto, colaborando y aportando cambios al código fuente. Cada miembro del equipo tiene su propia copia local del repositorio y pueden enviar y recibir cambios de otros miembros del equipo.

Colaboración remota:

permite a desarrolladores ubicados en diferentes lugares del mundo trabajar juntos en un mismo proyecto, sin necesidad de estar físicamente en la misma ubicación.

Contribución de la comunidad: en proyectos de código abierto

, cualquier persona puede colaborar en el proyecto a través de la presentación de parches, la corrección de errores, la implementación de nuevas funcionalidades, etc.

Integración continua:

se trata de un proceso automatizado que se encarga de verificar que los cambios realizados en el código no rompan la funcionalidad existente y que no afecten negativamente a la calidad del software.

Revisión de código:

consiste en revisar y analizar los cambios realizados por otro miembro del equipo antes de que se integren al repositorio principal. Esto permite detectar posibles errores o problemas antes de que lleguen al resto del equipo.

Gestión de problemas:

consiste en llevar un registro de los problemas, errores o mejoras que se deben realizar en el proyecto, asignar responsabilidades y hacer seguimiento del progreso.

Comentarios y discusiones:

en muchas plataformas de SCV, los miembros del equipo pueden comentar y discutir los cambios realizados, aportar sugerencias y hacer preguntas para aclarar dudas. Esto fomenta la comunicación y colaboración entre los miembros del equipo.