# Caso # 4

**Instrucciones:**

**Implemente el desarrollo de este documento utilizando la herramienta GIT para el control de versiones. Agregar al profesor a la lista de los pueden acceder con el correo** [**alexvg360@gmail.com**](mailto:alexvg360@gmail.com) **y entregar en el campus dirección para acceder a la información.**

**Conteste las siguientes preguntas:**

1. ¿Qué es control de versiones?

El control de versiones es un sistema que permite a los desarrolladores mantener un registro de los cambios realizados en el código fuente de un software a lo largo del tiempo. Este registro permite a los desarrolladores revertir cambios, comparar versiones anteriores, identificar quién realizó un cambio específico y trabajar en paralelo en diferentes versiones del mismo software.

Además, el control de versiones también facilita la colaboración y el trabajo en equipo en proyectos de software, ya que permite a los miembros del equipo ver y manejar versiones compartidas del código fuente.

Hay varios sistemas de control de versiones disponibles, como Git, SVN, Mercurial, etc. Todos ellos ofrecen características similares, pero cada uno tiene sus propias fortalezas y debilidades, por lo que es importante elegir el sistema adecuado para un proyecto en particular.

1. ¿Cuál es la importancia del tener un sistema de control de versiones?

El tener un sistema de control de versiones es importante por varias razones:

Seguimiento y gestión de cambios: Un sistema de control de versiones permite llevar un registro de los cambios realizados en el código fuente, lo que facilita la revisión y el seguimiento de los cambios.

Colaboración y trabajo en equipo: Un sistema de control de versiones permite a los miembros del equipo colaborar y trabajar juntos en el mismo código fuente, lo que puede mejorar la eficiencia y la velocidad de desarrollo.

Revertir cambios: Si algo sale mal después de realizar un cambio en el código fuente, un sistema de control de versiones permite revertir ese cambio y volver a una versión anterior que funcionaba correctamente.

Historial de cambios: Un sistema de control de versiones permite mantener un historial de los cambios realizados en el código fuente, lo que es útil para rastrear problemas y soluciones.

Versiones y ramas: Un sistema de control de versiones permite crear y manejar diferentes versiones del código fuente, lo que facilita el desarrollo de nuevas características y la resolución de problemas sin afectar a la versión estable del software.

1. ¿Qué implica un sistema de control de versiones?

Un sistema de control de versiones implica varios aspectos, incluyendo:

1. Repositorio: Un lugar centralizado para almacenar todas las versiones del código fuente.
2. Seguimiento de cambios: La capacidad de registrar y seguir todos los cambios realizados en el código fuente.
3. Versiones y ramas: La capacidad de crear y manejar diferentes versiones y ramas del código fuente.
4. Colaboración y trabajo en equipo: La capacidad de permitir a los miembros del equipo colaborar y trabajar juntos en el mismo código fuente.
5. Integración y integridad: La capacidad de integrar y asegurar la integridad de los cambios realizados por varios desarrolladores.
6. Revertir cambios: La capacidad de revertir cambios no deseados o revertir a una versión anterior del código fuente.
7. Historial de cambios: La capacidad de mantener un registro detallado del historial de cambios realizados en el código fuente.
8. Integración con herramientas de desarrollo: La capacidad de integrarse con otras herramientas de desarrollo, como los entornos de desarrollo integrado (IDEs), los sistemas de gestión de proyectos y los sistemas de prueba automatizados.

En resumen, un sistema de control de versiones es un conjunto de herramientas y prácticas que permiten llevar un registro de los cambios realizados en el código fuente, colaborar y trabajar en equipo, integrar cambios y revertir cambios no deseados.

1. ¿Cuáles son las ventajas de un tener un control de versiones?

Mejora de la colaboración y trabajo en equipo: Un sistema de control de versiones permite a los miembros del equipo colaborar y trabajar juntos en el mismo código fuente, lo que puede mejorar la eficiencia y la velocidad de desarrollo.

Mayor transparencia y seguimiento de cambios: Un sistema de control de versiones permite llevar un registro detallado de los cambios realizados en el código fuente, lo que facilita la revisión y el seguimiento de los cambios.

Mayor flexibilidad y capacidad de manejar versiones y ramas: Un sistema de control de versiones permite crear y manejar diferentes versiones y ramas del código fuente, lo que facilita el desarrollo de nuevas características y la resolución de problemas sin afectar a la versión estable del software.

Mayor seguridad y capacidad de revertir cambios: Un sistema de control de versiones permite revertir cambios no deseados o revertir a una versión anterior del código fuente, lo que puede mejorar la seguridad y la fiabilidad del software.

Mayor integridad y calidad del código: Un sistema de control de versiones permite integrar y asegurar la integridad de los cambios realizados por varios desarrolladores, lo que puede mejorar la calidad y la consistencia del código fuente.

Mejora de la productividad y eficiencia: Un sistema de control de versiones permite integrarse con otras herramientas de desarrollo, como los entornos de desarrollo integrado (IDEs), los sistemas de gestión de proyectos y los sistemas de prueba automatizados, lo que puede mejorar la productividad y la eficiencia del equipo de desarrollo.

1. ¿Cuál es el ciclo habitual de un sistema de control de versiones?

Clonar o descargar el repositorio: El primer paso es descargar o clonar el repositorio que se encuentra en el servidor de control de versiones a la máquina local del desarrollador.

Crear una rama: Antes de comenzar a realizar cambios en el código fuente, se crea una nueva rama para mantener los cambios separados de la rama principal o rama estable.

Realizar cambios y hacer commit: El desarrollador realiza los cambios necesarios en el código fuente y los guarda mediante el proceso de "hacer commit". Un commit es un punto de referencia en el historial de versiones que registra los cambios realizados en el código.

Hacer push: Una vez que los cambios han sido guardados en la máquina local, se hace "push" para publicarlos en el servidor de control de versiones.

Hacer merge: Cuando varios desarrolladores realizan cambios en el código fuente, es necesario combinar esos cambios en una única versión. Este proceso se conoce como "hacer merge".

Hacer pull request: Antes de que los cambios realizados en la rama sean integrados a la rama principal, es necesario hacer un "pull request". Un pull request es una solicitud de revisión y aprobación que se envía a otros miembros del equipo.

Aprobar y hacer merge: Si los cambios realizados en la rama son aprobados, se pueden combinar con la rama principal mediante el proceso de "hacer merge".

Revertir cambios: Si es necesario, se pueden revertir los cambios realizados en la rama principal, lo que permite corregir problemas o errores en el código.

En resumen, el ciclo habitual de un sistema de control de versiones implica descargar o clonar el repositorio, crear una rama, realizar cambios y hacer commit, hacer push, hacer merge, hacer pull request, aprobar y hacer merge, y revertir cambios si es necesario.

1. Cite algunas de las herramientas que existen para control de versiones

1. Git

2. Subversion (SVN)

3. Mercurial

4. Microsoft Team Foundation Server (TFS)

5. Apache AccuRev

6. Perforce

7. Bazaar

8. Visual SourceSafe (VSS)

9. SourceGear Vault

10. SourceAnywhere

1. ¿Cuál es la diferencia entre trabajar con bloqueo o sin bloqueo?

Cuando se trabaja con bloqueo, los usuarios deben "bloquear" un archivo antes de realizar cualquier cambio en él. Esto significa que otros usuarios no pueden modificar el archivo hasta que el usuario que lo ha bloqueado lo desbloquee. Esta configuración limita la cantidad de colisión de cambios que se pueden producir. Cuando se trabaja sin bloqueo, los usuarios no tienen que bloquear los archivos antes de realizar cambios en ellos. Esto significa que varios usuarios pueden realizar cambios en el mismo archivo al mismo tiempo, lo que aumenta el riesgo de colisión de cambios. Esto se soluciona generalmente agregando un mecanismo de control de versiones como Git, que permite detectar y resolver conflictos entre cambios realizados por varios usuarios.

1. ¿Para qué sirve el repositorio?

Es un lugar en el que se almacenan datos y código de manera organizada para que puedan ser compartidos y utilizados por varios usuarios. En él los usuarios contribuyen a los proyectos mediante el envío de cambios o revisiones al código. También se utiliza para almacenar y compartir información relacionada con la investigación científica, la publicación de libros y otros documentos, y para llevar a cabo la colaboración en línea.

1. ¿Qué es una copia de trabajo?

Una copia de trabajo es un conjunto de archivos y carpetas que están siendo gestionados por un sistema de control de versiones como Git. Esta copia de trabajo se crea cuando se clona un repositorio desde un servidor remoto, y se actualiza cada vez que se realizan cambios en el repositorio local. Estos cambios se pueden comunicar con el servidor remoto mediante operaciones como push y pull. La copia de trabajo incluye todos los archivos del repositorio, así como el historial de cambios, marcadores, ramificaciones y demás información necesaria para manejar el repositorio.

1. ¿Qué significa realizar un merge de los cambios y qué implica?

Realizar un merge de los cambios significa fusionar dos o más ramas en una sola. Esto se hace para combinar los cambios de una rama con los de otra. Por ejemplo, si se hacen cambios en la rama "rama1" y luego se hace un merge con la rama "rama2", los cambios de "rama1" se fusionarán con los de "rama2" para crear una nueva rama que contiene los cambios de ambas ramas. Esto se hace con el comando "git merge". El merge implica también la resolución de conflictos entre los cambios de las distintas ramas, de forma que los conflictos se deben resolver antes de que se pueda realizar el merge.

1. ¿Cómo trabajar en equipo con Git, sin morir en el intento?

Para trabajar en equipo con Git sin morir en el intento, es importante que todos los miembros del equipo estén familiarizados con el funcionamiento de Git y comprendan los conceptos básicos. Además, es importante establecer una estructura clara para el proyecto, que incluya una buena comunicación entre los miembros del equipo, así como también un sistema de control de cambios y un flujo de trabajo establecido. Además, es importante tener una buena documentación para facilitar el trabajo en equipo. Finalmente, es importante que todos los miembros del equipo estén al tanto de los cambios que se hagan para evitar conflictos y garantizar que todos los cambios sean visibles para el equipo.

**Elabore una pequeña encuesta, tome en cuenta algunos de las siguientes preguntas, queda a gusto del grupo tomarlas en cuenta o formular sus propias preguntas, no olvide agregar preguntas de tipo demográfico como edad, sexo y lugar de residencia por provincia.**

**¿Sabes lo que es un sistema control de versiones?**

(X ) Si

( ) No

**¿Sabes lo que es Github?**

( ) Se lo que es Github

( X) Se usarlo

( ) No se lo que es

( ) No se usarlo

**Que plataformas utiliza para guardar Código:**

( ) Dropbpx

( ) Drive

( X) GitHub

( ) Otro:\_\_\_\_\_\_\_\_\_\_\_\_\_

**¿Te parecería bien compartir tu código con todo el mundo y que puedan usarlo?**

( ) Compartir Código

(X ) No Compartir Código

**Una vez realizada la encuesta realice un informe de análisis de resultados.**