Instituto Politécnico Nacional
Escuela Superior de Comercio y Administración
Unidad Santo Tomás

Parc1_Pract1_Comparativa de software de versionamiento

Nombre: Ramírez Trejo Oswaldo
Asignatura. Laboratorio empresarial. Sistemas de información de gestión empresarial.
Profesor: Jovan Del Prado López
21/02/2025

ÍNDICE

RESUMEN

En esta práctica se analizan cinco sistemas de control de versiones utilizados en la industria: Git, Mercurial, Subversion (SVN), Bazaar

(Bzr) y Perforce Helix Core. Se investigaron aspectos clave como su historia, desarrollador principal, funciones destacadas, ventajas y desventajas, y su aplicación en distintos entornos. Además, se incluye una comparación entre estos sistemas. Durante la investigación se realizó un análisis de cada herramienta para conocer su funcionamiento. Se pudo observar cómo cada sistema maneja la gestión de archivos y cómo facilita el trabajo en equipo en entornos colaborativos. A partir de los resultados obtenidos, se analizaron los casos de uso de cada sistema para determinar cómo se puede utilizar en distintos tipos de proyectos.

INTRODUCCIÓN

El control de versiones es fundamental en el desarrollo de software, permitiendo gestionar cambios en el código y facilitar el trabajo en equipo. Esta práctica tiene como objetivo analizar cinco sistemas de control de versiones, identificando sus características y diferencias para determinar cuál es más adecuado según el tipo de proyecto. A medida que los proyectos de software crecen en complejidad, la necesidad de una herramienta eficaz de control de versiones se vuelve más evidente. Estas herramientas no solo permiten a los desarrolladores trabajar de manera independiente en diferentes partes del código sin interferir en el trabajo de otros, sino que también proporcionan un historial detallado de cambios, lo que facilita la detección y solución de errores. En esta práctica se analizará cada uno de los sistemas seleccionados, explicando sus capacidades, ventajas y desventajas, y evaluando su impacto en la productividad y eficiencia de los equipos de desarrollo. La investigación incluye ejemplos prácticos y flujos de trabajo típicos para demostrar cómo se utilizan estas herramientas en el mundo real.

DESCRIPCIÓN DE LA ENTIDAD

El trabajo se enfoca en herramientas de software utilizadas en empresas de tecnología, equipos de desarrollo y proyectos

colaborativos. Estas herramientas permiten a desarrolladores y equipos gestionar cambios en el código fuente de manera eficiente. Las empresas tecnológicas, en particular, dependen en gran medida de estos sistemas para garantizar que las versiones del software se gestionen correctamente, evitando la pérdida de datos y facilitando la colaboración entre distintos equipos. Un sistema de control de versiones también es clave para la automatización de procesos, permitiendo la integración continua y el despliegue automatizado de software. De esta manera, las organizaciones pueden mantener un ciclo de desarrollo rápido y confiable. Además, el control de versiones se extiende más allá del desarrollo de software, siendo una herramienta utilizada en la gestión de documentos, diseño gráfico, ingeniería y otras disciplinas que requieren colaboración en archivos digitales.

MARCO TEÓRICO

Los sistemas de control de versiones permiten registrar cambios en archivos a lo largo del tiempo. Se dividen en dos tipos:

- **Centralizados:** Un solo repositorio almacena el historial de cambios (ej. SVN, Perforce).
- **Distribuidos:** Cada usuario tiene una copia completa del repositorio (Git, Mercurial, Bazaar). Estos sistemas se utilizan para coordinar el trabajo entre varias personas y asegurar que los cambios en el código sean rastreables y revertibles si es necesario. En los sistemas centralizados, hay un servidor que gestiona el repositorio principal y los usuarios deben conectarse a este servidor para obtener las versiones más recientes de los archivos. En cambio, los sistemas distribuidos permiten a cada usuario tener una copia completa del historial, lo que proporciona mayor flexibilidad y resistencia ante fallos.

Actividades Realizadas

Se realizó una investigación detallada de los siguientes sistemas: Git, Mercurial, Subversion (SVN), Bazaar (Bzr) y Perforce Helix Core. Cada uno de estos sistemas fue analizado explorando sus características técnicas, ventajas y desventajas.

1. Git

Historia

Git fue creado en 2005 por Linus Torvalds, el mismo desarrollador del kernel de Linux. Antes de Git, el equipo de desarrollo de Linux utilizaba un sistema de control de versiones llamado BitKeeper, que tenía licencia privativa. Sin embargo, cuando BitKeeper dejó de ser gratuito para la comunidad de Linux, los desarrolladores se vieron en la necesidad de crear su propio sistema de control de versiones.

Torvalds diseñó Git con tres objetivos principales:

1. Velocidad: Debía ser rápido en todas sus operaciones.
2. Distribución: Cada desarrollador debía tener una copia completa del código y su historial.
3. Integridad de datos: Se debía garantizar que los archivos no fueran alterados sin ser detectados.

Desde su creación, Git se ha convertido en el sistema de control de versiones más utilizado en el mundo, siendo la base de plataformas como GitHub, GitLab y Bitbucket.

Desarrollador Principal

Inicialmente, Git fue desarrollado por Linus Torvalds, pero actualmente es mantenido por una gran comunidad de desarrolladores a través de GitHub y otros canales. Hay muchas empresas e individuos contribuyendo a su desarrollo, asegurando que el software evolucione y se mantenga actualizado.

Propósito del Software

Git es una herramienta diseñada para el control de versiones, lo que significa que ayuda a registrar los cambios en los archivos de un proyecto. Específicamente, permite a varios programadores trabajar en el mismo código al mismo tiempo sin que sus cambios interfieran entre sí.

Es especialmente útil en desarrollo de software, ya que:

- Guarda un historial de cambios en el código.
- Permite volver a versiones anteriores en caso de errores.
- Facilita la colaboración entre desarrolladores.
- Se integra con muchas plataformas y servicios para almacenamiento en la nube.

¿Centralizado o Distribuido?

Git es un sistema distribuido, lo que significa que cada usuario tiene una copia completa del código y su historial en su propia computadora. Esto tiene varias ventajas:
✓ No es necesario estar conectado a internet para hacer cambios y guardarlos.
✓ Se pueden hacer experimentos en el código sin afectar el repositorio central.
✓ Se evitan problemas si el servidor principal deja de funcionar.

Funciones Destacadas

1. Gestión de ramas (branches): Permite a los desarrolladores trabajar en diferentes características sin afectar el código principal.
2. Fusiones (merges): Facilita la combinación de cambios realizados en diferentes ramas.
3. Staging area: Un espacio intermedio donde los cambios pueden revisarse antes de confirmarlos.
4. Historial de versiones: Se pueden ver todos los cambios hechos en el proyecto y restaurar versiones anteriores si es necesario.

5. Colaboración en equipo: Integración con plataformas como GitHub, GitLab y Bitbucket para compartir código con otros desarrolladores.

Ventajas:

- Es rápido y eficiente.
- Permite trabajar sin conexión a internet.
- Es gratuito y de código abierto.
- Tiene una gran comunidad de soporte.

Desventajas:

- Puede ser difícil de aprender al inicio.
- No es tan intuitivo como otros sistemas.
- Algunos comandos pueden ser complicados.

Tipos de Proyectos Donde se Utiliza

Git se usa principalmente en desarrollo de software, pero también en otros proyectos como:

- Desarrollo web y aplicaciones móviles.
- Edición colaborativa de documentos.
- Gestión de configuraciones y scripts.

2. Mercurial

Historia

Mercurial fue creado en **2005** por **Matt Mackall** como una alternativa a BitKeeper y Git. Se diseñó para ser rápido y fácil de usar, especialmente en proyectos grandes.

Desarrollador Principal

El proyecto es mantenido por la comunidad de código abierto, con contribuciones de empresas como **Mozilla** y **Facebook**.

Propósito del Software

Mercurial es un sistema de control de versiones similar a Git, pero más simple de usar. Su objetivo es ofrecer una alternativa rápida y confiable para gestionar cambios en el código fuente.

¿Centralizado o Distribuido?

Es un **sistema distribuido**, lo que significa que cada usuario tiene una copia completa del código y su historial.

Funciones Destacadas

- **Interfaz más sencilla** que Git.
- **Mayor estabilidad en repositorios grandes.**
- **Mejor manejo de archivos binarios.**

Ventajas:

- Más fácil de aprender que Git.
- Es rápido en repositorios grandes.

Desventajas:

- Menos popular, por lo que tiene menos soporte.
- No tiene tantas integraciones como Git.

3. Subversion (SVN)

Historia

Desarrollado en el año 2000 por **CollabNet**, fue creado para reemplazar CVS, un sistema más antiguo.

Desarrollador Principal

Actualmente es mantenido por **Apache Software Foundation**.

Propósito del Software

SVN está diseñado para proyectos donde es importante mantener un historial centralizado de cambios.

¿Centralizado o Distribuido?

Es **centralizado**, lo que significa que depende de un servidor para almacenar la información.

Funciones Destacadas

- Control de acceso centralizado.
- Gestión de versiones con registros detallados.

Ventajas:

- Es más fácil de gestionar en grandes empresas.

Desventajas:

- No se puede trabajar sin conexión a internet.

4.Bazaar (Bzr)

Historia

Bazaar fue desarrollado por **Canonical** en **2005** con el objetivo de proporcionar un sistema de control de versiones flexible, fácil de usar y accesible para proyectos de código abierto y empresariales. Fue utilizado en el desarrollo de Ubuntu y otros proyectos relacionados con Linux. Sin embargo, Canonical dejó de mantenerlo activamente en 2017.

¿Centralizado o Distribuido?

- Bazaar es **híbrido**, lo que significa que puede **funcionar tanto como un sistema distribuido (como Git) o centralizado (como SVN)**, según las necesidades del equipo.
- Los desarrolladores pueden trabajar sin conexión y luego sincronizar con un servidor, o trabajar directamente en un repositorio central.

Propósito del Software

Bazaar fue diseñado para ofrecer una **experiencia intuitiva y flexible** en el control de versiones, especialmente para proyectos que requerían colaboración sencilla y compatibilidad con otros sistemas como Git y Subversion.

Funciones Destacadas

Integración con otros sistemas: Puede interactuar con Git y SVN.
Facilidad de uso: Tiene comandos intuitivos y una curva de aprendizaje menor que Git.
Flexibilidad en modelos de trabajo: Puede usarse como sistema distribuido o centralizado.

Historial completo en cada copia: Cada clon de un repositorio contiene todo el historial del proyecto.

Ventajas

Simplicidad: Tiene comandos más fáciles de entender que Git.
Modo dual: Puede operar en entornos centralizados o distribuidos.
Compatibilidad con otros sistemas: Puede trabajar con Git, SVN y otros.
Buena documentación y soporte en su momento: Aunque no es tan popular hoy en día, en su época tenía buen soporte.

Desventajas

Menos popularidad: Ha sido abandonado por Canonical y tiene poca comunidad activa.
Menor rendimiento en proyectos grandes: No es tan eficiente como Git en proyectos con muchos archivos.
Pocas herramientas modernas de integración: Al haber caído en desuso, tiene menos compatibilidad con plataformas actuales como GitHub o GitLab.

Tipos de proyectos donde se usa

- Proyectos de código abierto (principalmente en su época de auge).
- Desarrollo de software con equipos pequeños que necesitan flexibilidad.
- Proyectos híbridos que requieren compatibilidad con otros sistemas de control de versiones.

5 Perforce Helix Core

Historia

Perforce Helix Core fue desarrollado por Perforce Software en 1995. Se creó para empresas que necesitaban gestionar grandes volúmenes de código con equipos numerosos y requerimientos estrictos de control. Actualmente, es utilizado en industrias como los videojuegos, producción audiovisual y simulaciones debido a su capacidad de manejar archivos grandes y múltiples usuarios concurrentes.

¿Centralizado o Distribuido?

- **Principalmente centralizado**, lo que significa que todos los cambios se registran en un servidor central.
- Sin embargo, cuenta con opciones distribuidas para ciertos flujos de trabajo.

Propósito del Software

Su propósito es **ofrecer escalabilidad y control avanzado** en la gestión de código fuente y archivos grandes, especialmente en entornos empresariales con equipos grandes.

Funciones Destacadas

Gran escalabilidad: Puede manejar grandes cantidades de datos y archivos sin afectar el rendimiento.

Alta velocidad en operaciones grandes: Está optimizado para proyectos con miles de archivos.
Control de acceso avanzado: Permite administrar permisos de usuario de manera detallada.
Historial preciso y detallado: Registra cada cambio de forma organizada para auditoría y control.
Soporte para archivos binarios grandes: Ideal para videojuegos y modelos en 3D.

Ventajas

Rendimiento superior en grandes repositorios: No se ralentiza incluso con miles de archivos.

Control de acceso detallado: Permite gestionar permisos específicos para cada usuario.
Uso en la industria de videojuegos y cine: Empresas como Ubisoft, Pixar y EA lo utilizan.
Soporte empresarial: Tiene soporte técnico especializado para empresas.

Desventajas

No es Open Source: A diferencia de Git o Mercurial, Perforce es software privativo.
Curva de aprendizaje pronunciada: Su uso avanzado requiere capacitación.
Menos flexible que Git: No es tan adecuado para proyectos distribuidos o de código abierto.
Requiere infraestructura: Al ser centralizado, necesita servidores robustos para grandes equipos.

SOFTWARE	Características	Ventajas	Desventajas	paga / no paga
GIT	Es un sistema de control de versiones distribuido. Permite a cada usuario tener una copia completa del repositorio, lo que hace posible trabajar de manera autónoma.	**Desempeño rápido:** permite gestionar versiones de código de manera ágil sin necesidad de conexión a un servidor central, lo que optimiza el flujo de trabajo **Soporte amplio:** proporciona documentación, foros y herramientas adicionales para facilitar su uso **Flexible:** adaptar sus flujos de trabajo según sus necesidades, ya sea mediante ramas, bifurcaciones o integraciones en plataformas	**Curva de aprendizaje:** su uso puede ser complicado para principiantes debido a la variedad de comandos y opciones disponibles **Complejidad innecesaria para proyectos pequeños:** donde un control de versiones más sencillo podría ser suficiente. Esto puede generar una sobrecarga de procesos y herramientas que no siempre son requeridas en desarrollos de menor escala.	Es gratuito
Subversion (svn)	Es un sistema de control de versiones centralizado. Los usuarios deben conectarse al repositorio central para obtener los archivos y hacer cambios.	**Fácil de aprender:** su estructura de trabajo basada en un servidor central facilita la administración y el seguimiento de cambios, lo que resulta beneficioso para equipos que buscan una solución	**No permite trabajar offline:** depende de un servidor central para acceder a los archivos y realizar cambios. Esto puede ser un inconveniente en situaciones donde los desarrolladores	Es gratuito

		sencilla y estructurada **Control centralizado:** permite una gestión más ordenada y un mayor control sobre el acceso y las modificaciones realizadas por los usuarios	no tienen conexión a internet o cuando el servidor presenta fallos **Escalabilidad limitada:** puede volverse menos eficiente a medida que el proyecto crece y el número de archivos y usuarios aumenta	
Mercurial	Sistema de control de versiones, más sencillo en su diseño. Permite a cada usuario tener una copia completa del repositorio. Utiliza una interfaz de linea de comandos bastante amigable	**Fácil de usar:** su interfaz y comandos son más intuitivos, lo que facilita su adopción por parte de nuevos usuarios **Buena para proyectos medianos:** ofrece un rendimiento eficiente y estable sin la complejidad innecesaria de otros sistemas más avanzados. Su diseño busca minimizar errores y proporcionar un flujo de trabajo claro para los desarrolladores	**Menos soporte y herramientas:** puede dificultar su integración con ciertas plataformas y limitar las opciones para la gestión de proyectos **No tan robusto:** otros sistemas de control de versiones, especialmente en proyectos de gran escala donde la flexibilidad y la personalización son factores clave. Aunque es una alternativa confiable, en muchos casos los equipos prefieren herramientas con mayor soporte y comunidad activa	Es gratuito

Bazaar (Bzr)	Sistema de control de versiones distribuido Diseñado para ser fácil de usar y configurar. Tiene una integración estrecha con Ubuntu, aunque su uso ha disminuido con el tiempo.	**Fácil de aprender:** su diseño intuitivo y su sintaxis clara permiten que los usuarios lo adopten rápidamente sin necesidad de un aprendizaje complejo **Buena para proyectos pequeños:** ofrece una gestión eficiente de versiones sin la sobrecarga de funciones avanzadas que pueden ser innecesarias en desarrollos de menor escala	**No recomendado para grandes proyectos:** ya que su rendimiento puede verse afectado cuando se maneja una gran cantidad de archivos y cambios en el código	Es gratuito
Perforce Helix Core	Sistema de control de versiones centralizado, pero optimizado para rendimiento escalable. Maneja archivos binarios de manera eficiente, lo que lo hace adecuado para proyectos con grandes archivos.	**Alto rendimiento:** está optimizado para manejar grandes volúmenes de datos y cambios de manera rápida y eficiente **Ideal para archivos binarios grandes:** excelente opción para industrias como el desarrollo de videojuegos y la ingeniería de software, donde es común manejar archivos pesados que otros sistemas	**Licencia de pago para equipos grandes**: puede representar una barrera para startups o pequeños equipos de desarrollo que buscan soluciones gratuitas o de código abierto. Aunque ofrece versiones gratuitas para individuos o equipos pequeños, su implementación a gran escala implica costos que pueden ser elevados	Tiene una versión gratuita para equipos pequeños (hasta 5 usuarios) y versiones de pago para equipos grandes

		de control de versiones pueden gestionar con menos eficacia.		

Durante esta actividad descubrí diferentes tipos de control de versiones y sus aplicaciones según distintos contextos de trabajo.

Un aprendizaje clave fue la diferencia entre sistemas centralizados como Subversion y Perforce Helix Core, y sistemas distribuidos como Git, Mercurial y Bazaar. Git es la opción más popular por su flexibilidad e integración con plataformas modernas. Perforce se destaca en industrias como videojuegos y producción audiovisual debido a su capacidad para manejar archivos grandes.

Para proyectos de código abierto y desarrollo ágil, Git es la mejor opción. Subversion sigue siendo útil en entornos con estructuras jerárquicas, mientras que Perforce es ideal para grandes volúmenes de datos. Mercurial y Bazaar ofrecen alternativas más intuitivas para equipos pequeños, aunque Bazaar ha perdido soporte con el tiempo.

La elección del sistema de control de versiones depende del tamaño del equipo, el tipo de archivos y la infraestructura de la empresa. Comprender las necesidades específicas de cada proyecto es clave para seleccionar la herramienta adecuada.

CITAS

La arquitectura y la historia de Git: un sistema de control de versiones distribuido. (2020, 4 diciembre). ICHI.PRO. https://ichi.pro/es/la-arquitectura-y-la-historia-de-git-un-sistema-de-control-de-versiones-distribuido-10

Davidochobits. (2018, 15 noviembre). *Control de versiones distribuido con Mercurial - ochobitshacenunbyte*. Ochobitshacenunbyte. https://www.ochobitshacenunbyte.com/2016/05/20/control-de-versiones-distribuido-con-mercurial/

Subversión de Apache _ AcademiaLab. (s. f.). https://academia-lab.com/enciclopedia/subversion-de-apache/#google_vignette