**Sistema de control de versiones**

Cristhian Ferney Prieto Sánchez

Ronal Andrés Peña García

Brayan Andrés Velandia Rivera

Andrés Leonardo Salazar Mayorga

Tecnólogo en análisis y desarrollo de sistemas de información - Servicio nacional de aprendizaje SENA

2338322: Proyecto formativo

Ing. Hernando Enrique Moreno

22 de agosto de 2021

**Sistema de control de versiones**

Cristhian Ferney Prieto Sánchez

Ronal Andrés Peña García

Brayan Andrés Velandia Rivera

Andrés Leonardo Salazar Mayorga

Ing. Hernando Enrique Moreno

Servicio nacional de aprendizaje SENA

Tecnólogo en análisis y desarrollo de sistemas de información

Proyecto formativo

Bogotá D.C

2021

Contenido

[Introducción 5](#_Toc80560609)

[1. Sistema de control de versiones 6](#_Toc80560610)

[Ventajas de un sistema de control de versiones: 8](#_Toc80560611)

[Definir e identificar los diferentes tipos de sistemas de control de versiones 9](#_Toc80560612)

[CVS de versiones locales 9](#_Toc80560613)

[CVS Centralizado 10](#_Toc80560614)

[CVS Distribuidos 11](#_Toc80560615)

[1.3 Sistemas de control de versiones mas usados en el mercado 12](#_Toc80560616)

[Git 12](#_Toc80560617)

[CVS 12](#_Toc80560618)

[Mercurial 12](#_Toc80560619)

[Monotone: 12](#_Toc80560620)

[1.4 Las semánticas para manejo de versiones más usados en el mercado 13](#_Toc80560621)

[Versiona miento semántico 13](#_Toc80560622)

[X o major 13](#_Toc80560623)

[Y o minor 13](#_Toc80560624)

[Z o patch 13](#_Toc80560625)

[1.5 Buenas prácticas de calidad en manejo de sistema de control de versiones 14](#_Toc80560626)

[2. Principales sentencias para el manejo de sistema de control de versiones 15](#_Toc80560627)

[2.1 Semántica para versionar el proyecto formatico 15](#_Toc80560628)

[2.2 Sistema de calidad para el manejo de sistemas de control de versiones 15](#_Toc80560629)

[Conventional Commits 1.0.0 15](#_Toc80560630)

[Cuerpo 16](#_Toc80560631)

[Pie 17](#_Toc80560632)

[Técnica a usar. GIT FLOW 17](#_Toc80560633)

[2.3 Sistema de control de versiones para el repositorio del proyecto formatico 17](#_Toc80560634)

[3. Referencias 18](#_Toc80560635)

# Introducción

Mediante la evolución del proyecto, se busca afianzar y adquirir conocimientos para el desarrollo de un sistema de control de versiones teniendo en cuenta el reconocimiento teórico y práctico que se debe realizar a cada tipo de sistema para adoptar el más optimo y poner en practica su funcionamiento a lo largo del proyecto.

Teniendo en cuenta las bases teóricas aplicadas hasta el momento se espera poder reconocer y aplicar a profundidad las herramientas del sistema de control de versiones para cumplir con las metas y expectativas del proyecto.

# Sistema de control de versiones

El control de versiones, también conocido como “control de código fuente” es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo. A medida que los entornos de desarrollo se aceleran, los sistemas de control de versiones ayudan a los equipos de software a trabajar de forma más rápida e inteligente. Son especialmente útiles para los equipos de DevOps, ya que les ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas. El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

Para casi todos los proyectos de software, el código fuente es como las joyas de la corona, un activo valioso cuyo valor debe protegerse. Para la mayoría de los equipos de software, el código fuente es un repositorio del conocimiento de valor incalculable y de la comprensión sobre el dominio del problema que los desarrolladores han recopilado y perfeccionado con un esfuerzo cuidadoso.

El control de versiones protege el código fuente tanto de las catástrofes como del deterioro casual de los errores humanos y las consecuencias accidentales. Los desarrolladores de software que trabajan en equipos están escribiendo continuamente nuevo código fuente y cambiando el que ya existe. El código de un proyecto, una aplicación o un componente de software normalmente se organiza en una estructura de carpetas o “árbol de archivos”.

Un desarrollador del equipo podría estar trabajando en una nueva función mientras otro

desarrollador soluciona un error no relacionado cambiando código, cada desarrollador podría hacer sus cambios en varias partes del árbol de archivos.

El sistema de control de versiones ayuda a los equipos a resolver este tipo de problemas al realizar un seguimiento de todos los cambios individuales de cada colaborador y contribuir para evitar que el trabajo concurrente entre en conflicto. Los cambios realizados en una parte del software pueden ser incompatibles con los que ha hecho otro desarrollador que está trabajando al mismo tiempo. Este problema debería detectarse y solucionarse de manera ordenada sin bloquear el trabajo del resto del equipo. Además, en todo el desarrollo de software, cualquier cambio puede introducir nuevos errores por sí mismo y el nuevo software no es fiable hasta que se prueba. De este modo, las pruebas y el desarrollo van de la mano hasta que está lista una nueva versión.

Un buen sistema de software de control de versiones soporta el flujo de trabajo

preferido de un desarrollador sin imponer una forma determinada de trabajar. Idealmente, también funciona en cualquier plataforma, en vez de ordenar qué sistema operativo o cadena de herramientas deben utilizar los desarrolladores, Los sistemas de control de versiones excepcionales facilitan un flujo sencillo y continuo de cambios en el código en vez del mecanismo frustrante y burdo del bloqueo de archivos, que da facilita a un desarrollador a expensas de bloquear el progreso de los demás.

Los equipos de software que no utilizan ninguna forma de control de versiones a menudo se encuentran con problemas como no saber qué cambios que se han hecho están disponibles para los usuarios o la creación de cambios incompatibles entre dos partes no relacionadas que tienen que desvincularse y revisarse exhaustivamente. Si eres un desarrollador que nunca ha utilizado el control de versiones, puede que hayas añadido versiones a tus archivos, quizás con sufijos como “final” o “más”, y que después hayas tenido que enfrentarte con una nueva versión final. Quizás has convertido en comentarios bloques de código, porque quieres desactivar una determinada función sin eliminar el código, con el miedo de que pueda utilizarse más adelante. El control de versiones es una forma de solucionar estos problemas.

El software de control de versiones es una parte esencial del día a día de las prácticas profesionales del equipo de software moderno. Los desarrolladores de software individuales que están acostumbrados a trabajar con un sistema de control de versiones potente en sus

equipos suelen reconocer el increíble valor que el control de versiones también les da incluso en los proyectos pequeños en los que trabajan solos. Una vez acostumbrados a las potentes ventajas de los sistemas de control de versiones, muchos desarrolladores no se plantearían trabajar sin ellos incluso para los proyectos que no son de software.

# Ventajas de un sistema de control de versiones:

* Mantener una mayor eficiencia en los equipos de software de alto rendimiento ayudando a los desarrolladores a moverse más rápido dentro del sistema.
* Facilitar un historial de cambios a largo de plazo de los archivos
* Guardar registros y modificaciones realizadas a lo largo del tiempo por los desarrolladores incluyendo nombre, fecha y notas escritas sobre cada cambio
* Permitir el flujo de trabajo independiente y su vez facilitar la fusión de la información de cada desarrollador independiente
* Permitir a los desarrolladores hacer cambios correctos y armoniosos que estén en línea con el diseño previsto a largo plazo del sistema y garantizar a los desarrolladores calcular un trabajo futuro con precisión

# Definir e identificar los diferentes tipos de sistemas de control de versiones

Teniendo en cuenta la evolución de los sistemas de controles a través del tiempo, en la actualidad se pueden clasificar en 3 tipos: SISTEMAS DE CONTROL DE VERSIONES LOCALES, CENTRALIZADOS Y DISTRIBUIDOS.

CVS de versiones locales**:** en vez de mantener las versiones como archivos independientes, los almacenaban en una base de datos, a diferencia de las versiones anteriores en donde se tenían los archivos como directos y solo se tenia una copia del proyecto.

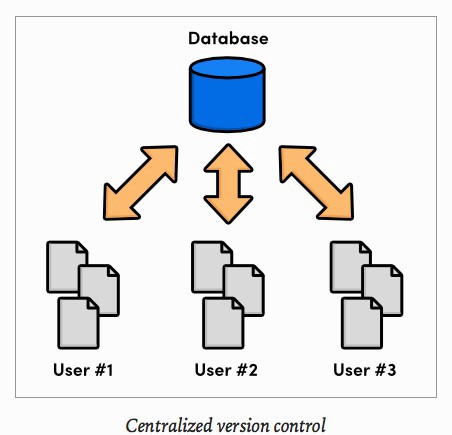
Este sistema de control se llevaba a cabo de manera independiente por cada desarrollador y no había manera de compartir el código entre sí.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

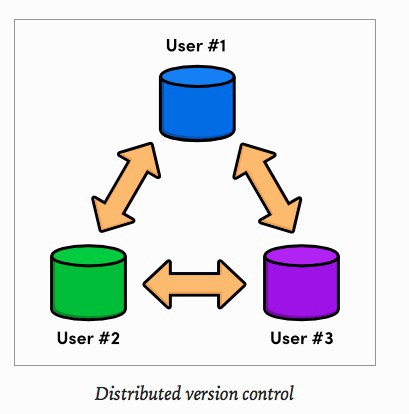
*Imagen: sistemas de control de versiones, que son y porque amarlos by JointDevoloper (junio 2017)*

CVS Centralizado**:** Facilitan la colaboración de múltiples desarrolladores para un solo proyecto y almacenan los cambios y versiones en un servidor. Bajo este avance, surge la incógnita de ¿Como varios desarrolladores podrían trabajar sobre un mismo proyecto a la vez?, sin embargo, los sistemas de versión de control centralizado mantienen un equilibrio para proyectos que requieran pocas actualizaciones y donde se pudiera mantener de acuerdo y en contexto de los posibles errores a todos los miembros desarrolladores del equipo.Pero hasta ahora no se encontró una alternativa para proyectos que resultaran con conflictos mas grandes



*Imagen: sistemas de control de versiones, que son y porque amarlos by JointDevoloper (junio 2017)*

CVS Distribuidos**:** Este sistema permite al desarrollador tener una copia local de todo el proyecto y elimino la idea de un repositorio centralizado garantizando al desarrollador que pudiera trabajar de manera aislada, pero asegurando una resolución de conflictos y posibles errores mas eficiente que la de versiones anteriores. Esto ayuda a tener el menor margen de riesgo de pérdida de información por una caída de servidor, algún tipo de daño o perdida de algún repositorio.



*Imagen: sistemas de control de versiones, que son y porque amarlos by JointDevoloper (junio 2017)*

# 1.3 Sistemas de control de versiones mas usados en el mercado

Git**:** es uno de los mejores sistemas de control de versiones disponible en el mercado actual. Es un modelo de repositorio distribuido compatible con sistemas y protocolos existentes como HTTP, FTP, SSH y es capaz de manejar eficientemente proyectos pequeños a grandes.

CVS**:** es un sistema de control de versiones muy popular. Es un modelo de repositorio cliente-servidor donde varios desarrolladores pueden trabajar en el mismo proyecto en paralelo.

El cliente CVS mantendrá actualizada la copia de trabajo del archivo y requiere intervención manual solo cuando ocurre un conflicto de edición. Apache Subversión (SVN): abreviado como SVN, es un modelo de repositorio cliente- servidor donde los directorios están versionados junto con las operaciones de copia, eliminación, movimiento y cambio de nombre.

Mercurial: es un sistema de control de versiones que está escrita en Python y destinada desarrolladores de software. Los sistemas operativos que admite son similares a Unix, Windows y macOS. Tiene un alto rendimiento y escalabilidad con capacidades avanzadas de ramificación y fusión y un desarrollo colaborativo totalmente distribuido. Además, posee una interfaz web integrada.

Monotone: está escrito leguaje en C++ y es un sistema de control de versiones

distribuido. El sistema operativo que admite incluye Unix, Linux, BSD, Mac OS X y Windows.

Brinda un buen apoyo para la internacionalización y localización. Además, utiliza un protocolo personalizado muy eficiente y robusto llamado Netsync.

*” (5 softwares de control de versiones – Drauta. Febrero 11, 2020. https://www.drauta.com/5-softwares-de-control-de-versiones)”*

# 1.4 Las semánticas para manejo de versiones más usados en el mercado

Versiona miento semántico: “Es una manera de identificar la versión de nuestro

software usando un patrón específico y así cualquiera que se interese en nuestro software y vea el historial de versiones vea cuánto se ha mejorado a través de cada una de las actualizaciones.

El número de la versión se identifica con tres dígitos, lo cual es un patrón X.Y.Z y cada posición tiene su significado:

X o major: Es cuando se realiza un cambio muy grande en el software, se borran o se añaden múltiples funcionalidades. Generalmente la versión anterior a ésta es incompatible con la nueva, por eso; al descargarla para el uso hay que tener consideraciones. Un ejemplo es la reciente versión 3.1.0 de jQuery, a comparación con la 2.2.4 hay muchas cosas que varían, quizá funciones nuevas o algunas removidas que pueden ocasionar incompatibilidad.

Y o minor: Es cuando se agrega una nueva funcionalidad en el software, pero esta sigue siendo compatible con la versión anterior. También es considerada cuando se marca algo del software como obsoleto.

Z o patch**:** Se utiliza cuando se corrigen bugs o fallas. No sólo se usa para corrección de funcionalidad, también es posible usar este dígito cuando se cambian aspectos estéticos. La compatibilidad con versiones anteriores se mantiene perfectamente

*.” (Versionamiento Semántico. Octubre 23, 2017. https://ernestoaguaysol.github.io/versionamiento-semantico/)*

# Buenas prácticas de calidad en manejo de sistema de control de versiones

1. “Si confirmas cambios de manera regular (a menudo) hace que las confirmaciones sean pequeñas y pueda confirmar solo cambios que estén relacionados.

2. Permite compartir el código con otros compañeros del equipo. De esa manera, será más

fácil para todos integrar los cambios regularmente y evitar tener conflictos al fusionar

cambios y ramas.

3. Si no se realizan confirmaciones con regularidad y se tienen pocas confirmaciones, pero

muy grandes, será difícil resolver los conflictos y comprender los cambios que se realizaron.

4. Crea mensajes de confirmación intuitivos y descriptivos es una de las mejores prácticas que

puede realizar, sobre todo para otras personas que utilicen el repositorio.

5. Las ramas son la herramienta perfecta para ayudar a evitar mezclar diferentes líneas de desarrollo. Debe usar ramas constantemente en sus flujos de trabajo de desarrollo: para nuevas funciones, correcciones de errores, experimentos, ideas, etc.

6. Actualiza tu repositorio antes de enviar cambios.

7. Divide el trabajo en repositorios” (Buenas prácticas en GIT: Confirme a menudo, use ramas,

8. “Analiza los cambios realizados por tus compañeros.

9. No hagas commit de algo roto.

10. Comenta siempre tus commits.

11. Realiza commits regularmente.

12. Comenta el código una vez resuelto el conflicto si has modificado su funcionalidad.

# 2. Principales sentencias para el manejo de sistema de control de versiones

# 2.1 Semántica para versionar el proyecto formatico

Especificación del Versionado Semántico (SemVer)

# 2.2 Sistema de calidad para el manejo de sistemas de control de versiones

# Conventional Commits 1.0.0

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

Antes de poner en práctica el sistema de calidad para el manejo de un sistema de control de versiones, y basados en el sistema escogido, se deben tener en cuenta los valores y estructura permitidos para este sistema:

**Valores permitidos**

**Feat:** Se utiliza cuando se está confirmando una nueva característica estable, es decir que es algo que no se ha creado antes.

**Fix:** Representa la corrección de una falla(bug) dentro del proyecto.

**Docs:** Representa cambios en la documentación del proyecto.

**Style:** Representa cambios en el estilo (interfaz) del proyecto.

**Refactor:** Representa modificaciones y cambios para optimizar y mejorar el diseño y/o arquitectura.

**Test:**  Representa cambios o creación en pruebas del proyecto (unitarias, integración, funcionales, aceptación, …)

**Chore:** Representa cambios o creación en archivos que no afectan funcionalidades programáticas del sistema.

# Cuerpo

El cuerpo del mensaje de confirmación deberá dar información de utilidad y concisa que ayude a determinar en un vistazo de que se trata el cambio que se está llevando a cabo de modo que no de

lugares a dudas. Normalmente se suele escribir y detallar de forma genérica el cómo se resolvió sin entrar en detalles tan técnicos.

Sin embargo, en caso de usar un sistema de planeación de información que genere identificadores de “issues” (Redmine, bugtrack, Github, Gitlab, Mantis, etc…) es preferible apuntar al identificador del “issue” para evitar la repetición de información y documentación. Siendo que la documentación detallada será aplicada en el sistema de planificación y rastreo. Si la confirmación corresponde a un issue con un solo identificador entonces es posible representar el commit.

# Pie

Si existe un cambio que es muy peligroso y que puede dañar diversas características (no debería si existe ortogonalidad y correcto versionado semántico) es requerido que se coloque en el pie de la confirmación, el cual consiste en una línea en blanco después del cuerpo del mensaje.

# Técnica a usar. GIT FLOW

-Rama de producción: Main

-Rama de desarrollo: Develop

-Rama de requerimiento: Feacture/(nombre del requerimiento)

-Rama de solución de errores: hotfix

-Rama de lanzamiento: Releace/0.0.0

# 2.3 Sistema de control de versiones para el repositorio del proyecto formatico

**Link repositorio.:**  https://github.com/SENA-CEET/2338322-documents-2303322-grupo03

# Referencias

*” (Pérez Arques, A. (2009, Noviembre 10). Buenas prácticas de uso de un sistema de control de*

*versiones (i) [Web log post]. Blog de Julio César Pérez Arques.*

*http://jcesarperez.blogspot.com/2008/11/buenas-prcticas-de-uso-de-un-sistema-de.html)*

*divida y venza. 06/03/2019. https://revistadigital.inesem.es/informatica-y-tics/git/)*

5 softwares de control de versiones – Drauta. Febrero 11, 2020.

https://www.drauta.com/5-softwares-de-control-de-versiones

Versionamiento Semántico. Octubre 23, 2017.

https://ernestoaguaysol.github.io/versionamiento-semantico/

Buenas prácticas en GIT: Confirme a menudo, use ramas, divida y venza.

06/03/2019. https://revistadigital.inesem.es/informatica-y-tics/git/

Pérez Arques, A. (2009, Noviembre 10). Buenas prácticas de uso