# Sistema de Control de Versiones.

Guillermo Daniel Cruz Ortega
guillermo.daniel.cruz.ortega@gmail.com
Universidad de la Sierra Sur.

2022/05/09

## 1 Introducción.

Los sistemas de control de versiones tal como se conoce en el area de creación de software empezo a implementarse en la decada de los noventa, sin embargo estos en sus incios eran primitivos y muy limitados en funcionalidad, pero como todo en la tecnologia con el paso del tiempo la utilidad y mejoras alcanzaron esta area, perimitiendo tener copias de seguridad en grandes cantidades, y desde cualquier lugar, y no solo eso si no que la capacidad de escoger el sistenma de control de versiones a aumentado, a tal punto de que puede haber dos gestores completamente diferentes. Estos gestores permiten un amplio trabajo en equipo, perimtiendo unir a la gente a pesar de la distancia fisica que los separa y tambien el poder volver de forma segura a alguna version estable del programa, aumentando la productividad y la facilidad de movilizarse y seguir trabajando. EL control de versiones no solo se ocupa en progrmas o software, igual se ocupa en otras areas de la tecnologia o de las lineas de fabricación, permaneciendo los productos anteriores como referencia ante algun fallo o mejorar que se quiera realizar a futuro.

## 2 Control de Versiones

"Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o VCS (del inglés Version Control System). Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico). Ejemplos de este tipo de herramientas son entre otros: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Plastic SCM, Git, SCCS, Mercurial, Perforce, Fossil SCM, Team Foundation Server.

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente dando lugar a los sistemas de control de código fuente o SCM (siglas del inglés Source Code Management). Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etc." (*Wikipedia*, 2022)

## 3 Tipos de Control de Versiones

"Podemos clasificar los sistemas de control de versiones atendiendo a la arquitectura utilizada para el almacenamiento del código: locales, centralizados y distribuidos.

- Locales: Los cambios son guardados localmente y no se comparten con nadie. Esta arquitectura es la antecesora de las dos siguientes. 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 ingeniosos). Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías. Una de las herramientas de control de

versiones más popular fue un sistema llamado RCS, que todavía podemos encontrar en muchas de las computadoras actuales. Incluso el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo. Esta herramienta funciona guardando conjuntos de parches (es decir, las diferencias entre archivos) en un formato especial en disco, y es capaz de recrear cómo era un archivo en cualquier momento a partir de dichos parches.

- Centralizados: Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS y Subversion. Un gran problema con el que se encuentran las personas es que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar este problema. Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años. Esta configuración ofrece muchas ventajas, especialmente frente a VCS locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente. Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.

- Distribuidos: Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial. Los sistemas de Control de Versiones Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos. Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos." (*Wikipedia Apache Ant*, 2021)

# 4 Caracteristicas

"Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación...).

- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).

- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

Aunque no es estrictamente necesario, suele ser muy útil la generación de informes con los cambios introducidos entre dos versiones, informes de estado, marcado con nombre identificativo de la versión de un conjunto de ficheros, etc." (*Wikipedia Apache Ant*, 2021)

# 5 Repositorio de Código Fuente

"El control de código fuente (o el control de la versión) es la práctica de seguimiento y administración de los cambios en el código. Los sistemas de administración de código fuente (SCM) mantienen un historial del desarrollo de código, y ayudan a resolver conflictos a la hora de combinar las contribuciones de diferentes orígenes. Entre sus bveneficios estan:

- Permite el trabajo en paralelo de dos o más usuarios de una aplicación o programa sin que se den por válidas versiones con elementos que entren en conflicto entre sí.
- La seguridad de un repositorio de código en un servidor es máxima ya que este garantiza la misma mediante diferentes métodos avanzados de ciberseguridad y la creación constante de copias de seguridad.
- Permite disponer y acceder a un historial de cambios. Cada programador tiene la obligación de señalar qué cambios ha realizado, quién los ha llevado a cabo y también cuando se hicieron. Esto permite un mayor control sobre cada versión, permite corregir cambios y facilita que cada cambio realizado se vea como un nuevo estado.
- Facilita el entendimiento del proyecto, sus avances y el estado actual de la última versión. Algo posible gracias a que cada pequeño cambio realizado por un programador debe ir acompañado por un mensaje explicativo de la tarea realizada. Así evita a los demás programadores perder tiempo cuestionándose el por qué de los cambios realizados. Y facilita la corrección de errores si los hubiera y son detectados por otro programador.

# 6 Tipos de Repositorios de Codigo Fuente.

Git: Utiliza el control de versiones distribuido, dispone de copias locales del repositorio de código en las que los programadores trabajan directamente, cada repositorio así como cada copia de trabajo individual incluye el historial completo de cambios realizados en la aplicación o programa, es un sistema multiplataforma que se puede usar a través de la línea de comando o con múltiples clientes, es compatible con Github y Microsoft Visual Studio Code, solo precisa la conectividad a la red para la sincronización de cambios, etc.

SVN: Dispone de control de versiones centralizado, se basa en un repositorio central en el que se generan copias de trabajo para los programadores, solo permite acceder al historial de los cambios en el repositorio completo. Las copias de trabajo solo incluyen la última versión, es decir, la más reciente, el sistema de creación de ramas y etiquetas es muy eficiente, el seguimiento de los cambios se basa en archivos, etc. "(*Amazon*, 2020)

# 7 Arquitecturas de Almacenamiento.

"Podemos clasificar los sistemas de control de versiones atendiendo a la arquitectura utilizada para el almacenamiento del código:

- Distribuidos: cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.
- Centralizados: existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS, Subversion o Team Foundation Server.

Ventajas de sistemas distribuidos: Al hacer de los distintos repositorio una réplica local de la información de los repositorios remotos a los que se conectan, la información está muy replicada y por tanto el sistema tiene menos problemas en recuperarse si por ejemplo se quema la máquina que tiene el repositorio remoto. Por tanto hay menos necesidad de backups. Sin embargo, los backups siguen siendo necesarios para resolver

situaciones en las que cierta información todavía no haya sido replicada, permite mantener repositorios centrales más limpios en el sentido de que un usuario puede decidir que ciertos cambios realizados por él en el repositorio local, no son relevantes para el resto de usuarios y por tanto no permite que esa información sea accesible de forma pública, ya sea porque contiene versiones inestables, en proceso de codificación o con etiquetas personalizadas y de uso exclusivo del usuario, etc.

Ventajas de sistemas centralizados: En los sistemas distribuidos hay menos control a la hora de trabajar en equipo ya que no se tiene una versión centralizada de todo lo que se está haciendo en el proyecto. En los sistemas centralizados las versiones vienen identificadas por un número de versión. Sin embargo en los sistemas de control de versiones distribuidos no hay números de versión, ya que cada repositorio tendría sus propios números de revisión dependiendo de los cambios. En lugar de eso cada versión tiene un identificador al que se le puede asociar una etiqueta (tag)." (*Wikipedia*, 2020)

# 8 Conclusiones

Los sistemas de control de versiones han resultado muy útiles a lo largo de la historia del desarrollo de software, permitiendo que el software este mejor desarrollado. Antes estos sistemas de versiones estaban reservados para solo empresas de desarrollo, pero en la actualidad cualquier usuario puede acceder a uno de estos sistemas de control de versiones, sin embargo sigue habiendo funciones que se pueden adquirir por medio de paga, un ejemplo de esto es el espacio que permite almacenar en estos repositorios, igual el que tu proyecto o trabajo pueda ser publico permite aumentar la colaboración y el avance y en caso de que lleguen a borrar tu código, tendrás las copias de seguridad que hallas elaborado con anterioridad.

# 9 Referencias.

Control de versiones. (2022, 29 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 02:49, mayo 10, 2022 desde https://es.wikipedia.org/w/index.php?title=Controldeversionesoldid=143208812.

Sergio Gómez . (s. f.). Sistemas de control de versiones - Taller de Git. Taller de Git. Recuperado 9 de mayo de 2022, de https://aulasoftwarelibre.github.io/taller-de-git/cvs/

¿Qué es el control de código fuente? - Amazon Web Services. (s. f.). Amazon Web Services, Inc. Recuperado 9 de mayo de 2022, de https://aws.amazon.com/es/devops/source-control/

The Black Box Lab. (2021, 22 febrero). Qué son los repositorios de código y cuáles son sus beneficios. Recuperado 9 de mayo de 2022, de https://theblackboxlab.com/2021/02/22/que-son-los-repositorios-de-codigo-y-cuales-son-sus-beneficios/

Git - Acerca del Control de Versiones. (s. f.). Git-Scm. Recuperado 9 de mayo de 2022, de https://git-scm.com/book/es/v2/Inicio—Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones