INSTITUTO TECNOLOGICO DE CHIHUAHUA II

Documento

“Sistemas de Control de Versiones “CVS”

Ingeniería Informática.

Ingeniería en Sistemas Computacionales

Joan Armando Luna Sosa B16060106

Paulina Trevizo Castillo 18550322

Eder Antonio Hernández Navarrete - 18550736

Sofia Iveth Sinaloa García - 19550726

Joel Enrique Ortega Torres 19550367

Maestro: Carlos Humberto Rubio Rascón

Chihuahua, Chih. A 28 de febrero del 2022

Contenido

Introducción

Cuando se desarrolla código dentro de un equipo pueden generarse una variedad de problemas, dentro del equipo puede haber problemas desde no poder controlar la manera en que se trabaja en equipo, no encontrar un buen ritmo de trabajo con todos los involucrados hasta los posibles conflictos que haya creativamente en el entorno.

Cuando se involucra de lleno el trabajo en el código hay un sinfín de problemas con los que lidiar; el código principal llegue a estar corrupto, se pierda o sea imposible reemplazarlo. Al trabajar directamente de un código principal se corre el riesgo que los cambios efectuados no funcionen y no sea posible regresar al código posterior al cambio debido a que no se hizo o se hace respaldos periódicamente del mismo.

El desarrollo de software se basa en constantes cambios, análisis por un equipo variado, correcciones de errores, y a la larga genera una evolución social y creativa.

La programación por sí sola tiende a ser caótica, uniendo a la fórmula un grupo de personas, con diferentes tipos de ideas, sin preparación ni coordinación sólo puede llevar a un caos.

Determinando que el trabajo simultáneo es una necesidad y ese mismo trabajo simultáneo debe estar bien coordinado, saber quién y cómo realizó los cambios pertinentes es necesario metodologías o sistemas que ayuden a que el equipo desarrollador evite el mayor error humano posible. Si el trabajo, cambios, correcciones, resultan ser contraproducentes tener una forma de dar un paso atrás y eliminar todo aquel trabajo contraproducente.

Dentro de cualquier trabajo profesional se lleva una realización de gestión, los trabajos elaborados por desarrolladores de software no son la excepción; siendo una necesidad para poder identificar errores e introducir cambios y nuevos trabajos a un proyecto se usan herramientas para la realización de todas estas actividades. Una de ellas son los sistemas de control de versiones.

El sistema de control de versiones es una herramienta que se encarga de la gestión de modificaciones realizadas en archivos de texto, una ventaja para aquellos que trabajen con código en distintas versiones de archivos.

Para los equipos de trabajo “CVS” resulta ser la mejor solución a los variantes errores que pueden presentarse tanto con la arquitectura del mismo código como con el error humano.

En el siguiente trabajo se presenta que son los CVS de una forma mucho más extendida, junto con sus características primordiales y la presentación de un ejemplo de CVS.

CVS

Por sus siglas en inglés “Concurrent Versions System” es un Sistema de Control de Versiones, mantiene el registro de todo el trabajo y los cambios en los archivos de código fuente que forman parte de un proyecto de software. Un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueda recuperar versiones específicas más adelante.

Permite restaurar los archivos a un estado anterior, restaura todo el proyecto a un estado anterior, comparará los cambios a lo largo del tiempo y, por último, solucionará el problema potencial, quién introdujo el error, puede verificar la hora, etc.

Un buen software de control de versiones admite el flujo de trabajo preferido por los desarrolladores sin forzar una forma específica de trabajo. Idealmente, en lugar de dictar qué sistema operativo o desarrollador de cadena de herramientas usar, funciona en cualquier plataforma. Un buen sistema de control de versiones promueve un flujo suave y continuo de cambios de código en lugar del mecanismo sofocante y tosco de bloqueo de archivos. Esto permite que un desarrollador obtenga la aprobación a cambio de interrumpir el progreso de otro desarrollador.

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 reciente", y que después hayas tenido que enfrentarse 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.

Características.

- Utiliza una arquitectura cliente-servidor. El servidor almacena la versión actual del proyecto y su historial. El cliente se conecta al servidor y obtiene una copia completa del proyecto. Esto se hace para que finalmente pueda trabajar en esa copia para que pueda confirmar sus cambios más tarde usando los comandos GNU.
- El sistema CVS tiene la tarea de mantener el registro de la historia de las versiones del programa de un proyecto solamente con desarrolladores locales.
- Varios clientes pueden sacar una copia de un proyecto al mismo tiempo. Si la corrección se actualiza más tarde, el servidor intentará vincular otra versión.

- Los clientes pueden comparar diferentes versiones de un archivo, solicitar un historial de cambios completo o una "foto" del historial del proyecto para una fecha específica o un número de revisión.
- El cliente también puede usar el comando de actualización para actualizar la réplica a la última versión en el servidor. Esto elimina la necesidad de descargar repetidamente todo el proyecto.
- CVS le permite mantener varias "ramas" de su proyecto. Esto está actualmente en desarrollo y se puede hacer mientras una versión con cambios importantes con nuevas funciones está en otra fila formando otra rama.

Ventajas

- Un historial completo de cambios a largo plazo de todos los archivos. Esto significa todos los cambios realizados por muchas personas a lo largo de los años. Los cambios incluyen la creación y eliminación de archivos, así como cambios en su contenido. Las diferentes herramientas de VCS difieren en lo bien que manejan el cambio de nombre y el movimiento de archivos. Este historial también debe incluir el autor, la fecha y las notas escritas sobre el propósito de cada cambio. Tener el historial completo le permite retroceder a versiones anteriores para ayudar a analizar la causa raíz de los errores y es crucial al solucionar problemas de versiones anteriores de software. Si está trabajando activamente en el software, casi todo puede requerir una "versión anterior" del software.
- Creación de sucursales y fusiones. Si tiene miembros del equipo trabajando al mismo tiempo, no hay que pensarlo dos veces; pero incluso las personas que trabajan solas pueden beneficiarse de la capacidad de trabajar en flujos de cambio independientes. La creación de una "sucursal" en las herramientas de VCS mantiene múltiples flujos de trabajo independientes entre sí, al tiempo que brinda la capacidad de fusionarse nuevamente en ese trabajo, lo que permite a los desarrolladores verificar que los cambios de cada sucursal no hayan cambiado. ellos entran en conflicto. Muchos equipos de software adoptan la práctica de crear ramas para cada característica o quizás para cada versión, o para ambas. Hay muchos flujos de trabajo diferentes entre los que los equipos pueden elegir al decidir cómo usar la bifurcación y la fusión en VCS.
- Trazabilidad. Ser capaz de realizar un seguimiento de cada cambio que se realiza en el software y conectarlo a un software de seguimiento de errores y gestión de proyectos como Jira, además de poder anotar cada cambio con un mensaje que describe el propósito y el objetivo del cambio, no solo qué para ayudar con el análisis de causa raíz y la recopilación de información. Tener el historial anotado del código al alcance de su mano cuando está leyendo el

código, tratando de entender qué hace y por qué está diseñado de la manera que está, puede permitir a los desarrolladores realizar cambios correctos y armoniosos que están en línea a largo plazo. proporcionó. -Término de diseño de código. sistema. Esto puede ser especialmente importante para trabajar de manera eficiente con código heredado y es esencial para que los desarrolladores puedan cronometrar con precisión el trabajo futuro.

Desventajas.

- Los archivos en el repositorio de la plataforma CVS no se pueden renombrar, deben agregarse con otro nombre y luego eliminarse.
- El protocolo CVS no proporciona una forma de eliminar o cambiar el nombre de los directorios, cada archivo en cada subdirectorio debe eliminarse y volver a agregarse con el nuevo nombre.
- Soporte limitado para archivos Unicode con nombres de archivo que no sean ASCII.

Arquitecturas de almacenamiento.

Sistema de control de versiones locales.

El método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son avispados). Esta técnica es muy común porque es muy simple, pero también tremendamente propensa a cometer errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos.

Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco.

Sistemas de control de versiones centralizados

El siguiente gran problema con el que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solucionar este problema, se desarrollaron sistemas de control de versiones centralizados.

Esta configuración ofrece muchas ventajas, especialmente sobre el VCS local. Por ejemplo, todos saben hasta cierto punto en qué están trabajando los demás en el proyecto. Los administradores tienen un control detallado sobre lo que cada uno puede hacer.

Sin embargo, esta configuración también tiene serios inconvenientes: Si ese servidor se cae durante una hora, entonces nadie puede colaborar o guardar cambios versionados en lo que están trabajando durante esa hora. Si el disco duro donde se encuentra la base de datos central se corrompe y no se han realizado las copias de seguridad adecuadas, se pierde absolutamente todo el historial del proyecto, excepto aquellos momentos que las personas puedan tener en sus máquinas locales.

Sistemas de control de versiones distribuidos

Los clientes no sólo descargan la última versión de los archivos: replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos.

Es más, muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente de maneras distintas simultáneamente dentro del mismo proyecto. Esto te permite establecer varios tipos de flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

Términos relacionados.

Repositorio: Conjunto de código y el historial de los cambios realizados sobre el código. Lugar en el que se encuentran almacenados los archivos con sus respectivas versiones.

Ramas(Branches): Árboles de código que se independizaron del central, pero que comparten una historia común.

Trunk: Contiene la última versión del proyecto que se está desarrollando, la versión común a todos los colaboradores.

Tag: Copias de seguridad (Snapshots) del proyecto. A diferencia de los branches, lostags no permiten modificar archivos.

Parches: Modificaciones realizadas al código.

Merge: Mecanismo usado para fusionar branches.

Tipos de sistemas de control de versiones.

- **GIT:** Sistema de control de versiones gratuito y de código abierto, creado originalmente por Linus Torvalds en 2005. A diferencia de los antiguos sistemas centralizados de control de versiones, como SVN y CVS, Git está distribuido: cada desarrollador tiene el historial completo de su repositorio de código de manera local. De este modo, la clonación inicial del repositorio es más lenta, pero las operaciones posteriores, como commit, blame, diff, merge y log son mucho más rápidas.

    - *git init:* Este comando creará una carpeta .git en la que se encuentra el repositorio local. En este momento los archivos en la carpeta no están gestionados por el sistema de gestión de versiones.
    - *git add:* En donde se reemplazará por el nombre del archivo. En el caso de que se deseen incluir todos los archivos de la carpeta no es necesario realizar la tarea a mano. Simplemente se ha de utilizar punto (.) en lugar del nombre del archivo en el comando.
    - *git commit:* Cada una de las subidas al repositorio ha de contener un mensaje que permita identificar los cambios que contienen. Para esto aparecerá un editor de texto en el que se pedirá que se introduzca la descripción de la subida. Opcionalmente se le puede indicar el mensaje en el propio comando mediante la opción -m. Para lo que se ha de entrecomillar en mensaje inmediatamente después del comando.
    - *git status:* El cual dará como resultado el listado de archivos con cambios. Para agregar estos primero se han de añadir con el comando add que ya se ha visto o con el modificador -a en comando commit. Para agregar los cambios en un único paso simplemente se ha de escribir el comando
    - *git commit -a -m “Cambios”:* Consiguiente que se añadan todos los cambios a una nueva versión que contiene el mensaje “Cambios”.

Obviamente este mensaje no es significativo, siendo necesario explicar más los cambios en un repositorio real.

- *git remote add:* En esta ocasión se ha de reemplazar por un nombre para el repositorio y con la ubicación de este. Una vez indicado el repositorio se puede ver la lista de repositorios remotos mediante el comando
- *git remote:* Para recibir los cambios se ha de escribir
- *git fetch:* Por otro lado, para enviarlos se ha de utilizar
- *git push:* En ambos casos es el nombre que se le ha dado previamente al repositorio y es el nombre de la rama local que se desea sincronizar.

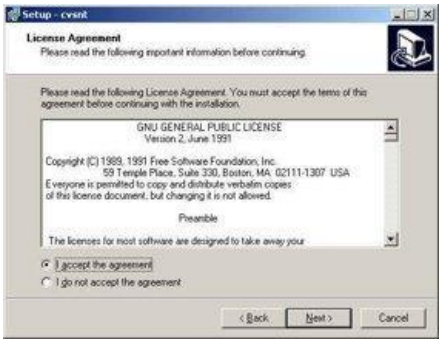
- **Subversion**: Subversion fue escrito ante todo para reemplazar a CVS—es decir, para acceder al control de versiones aproximadamente de la misma manera que CVS lo hace, pero sin los problemas o falta de utilidades que más frecuentemente molestan a los usuarios de CVS. Uno de los objetivos de Subversion es encontrar la transición a Subversión relativamente suave para la gente que ya está acostumbrada a CVS.

- **SVK:** Aunque se ha construido sobre Subversión, probablemente SVK se parece más a algunos de los anteriores sistemas descentralizados que a Subversión. SVK soporta desarrollo distribuido, cambios locales, mezcla sofisticada de cambios, y la habilidad de ""reflejar/clonar"" árboles desde sistemas de control de versiones que no son SVK.

- **Mercurial:** Mercurial es un sistema de control de versiones distribuido que ofrece, entre otras cosas, una completa "indexación cruzada" de ficheros y conjuntos de cambios; unos protocolos de sincronización SSH y HTTP eficientes respecto al uso de CPU y ancho de banda.

Instalación.

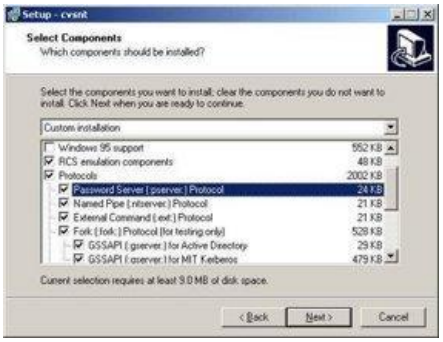
1. Abrimos el instalador y aceptamos la licencia.



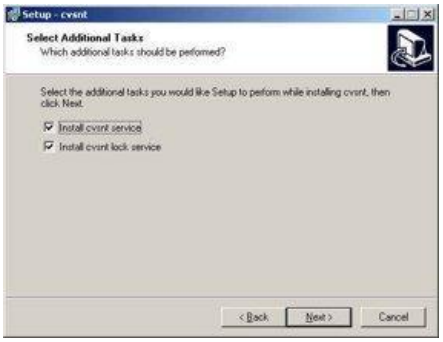
2. Elegimos la carpeta de instalación.

3. Desactivar el antivirus para evitar interferencia en la instalación.

4. Seleccionamos los componentes a instalar.



5. Seleccionamos instalar cvsnt como servicio y el cvsnt lock como servicio.

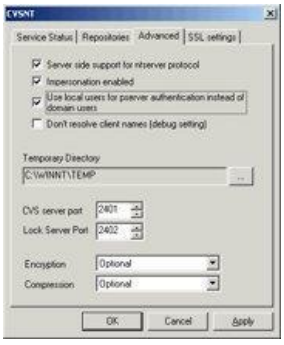


6. De esta manera tenemos instalado el servidor .

7. Abrimos la configuración en el panel de control, donde, nos permite activar o desactivarlo.

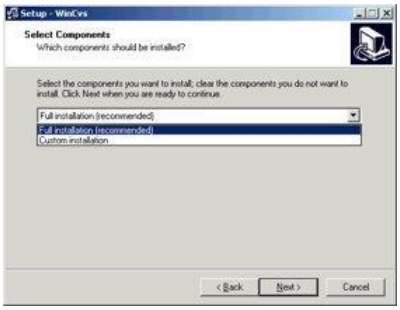
8. Seleccionamos la opción Repositories donde vamos a configurar las carpetas donde tendremos los desarrollos de software.

9. En advanced podemos configurar varias opciones, en este caso seleccionamos que se utilizarán usuarios locales en lugar de usuarios de dominio.



10. Para utilizar el cvs, se necesita un software cliente para cada desarrollador, en este caso utilizaremos el wincvs el cual se descarga desde esta url http://cvsgui.sourceforge.net/download.html
11. Abrimos el instalador y aceptamos el acuerdo de licencia.

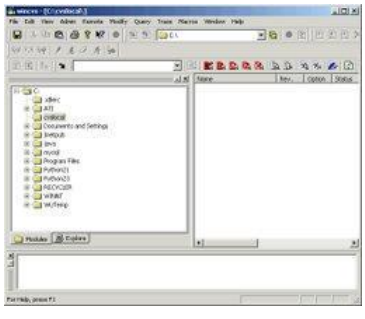
12. Elegimos la instalación completa.



13. Por último, finalizamos la instalación y marcamos la casilla para abrir el cliente.

14. Enseguida nos solicita que instalemos python, debemos instalarlo desde https://www.python.org

15. Una vez instalado el cliente wincvs, vemos la siguiente interfaz.



Conclusión

Como primera instancia se puede concluir que CVS es una herramienta poderosa y sencilla para el uso de grandes proyectos, en donde interactúan una gran cantidad de usuarios. Al finalizar este trabajo se puede decir que se ha adquirido los conceptos fundamentales acerca de su trabajo se han aprendido aspectos de instalación, configuración, creación y manipulación de sistemas de control de versiones. Es importante recordar que esta presentación cubre los aspectos fundamentales de CVS; para aquellos usuarios que quieran profundizar más en el tema, deben dirigirse a las referencias.

Bibliografía

A. Sistemas de Control de Versiones Libres. (s. f.). . https://producingoss.com/es/vc-systems.html

Atlassian. (s. f.). *GIT*. https://www.atlassian.com/es/git

uniwebsidad. (s. f.). *Acerca del control de versiones*. https://uniwebsidad.com/libros/pro-git/capitulo-1/acerca-del-control-de-versiones#:%7E:text=Sistemas%20de%20control%20de%20versiones,tremendamente%20propensa%20a%20cometer%20errores.

Comparación de las ventajas y desventajas de varios sistemas de control de versiones de uso común - programador clic. (s. f.). . https://programmerclick.com/article/99781868840/

Atlassian. (s. f.-b). *Qué es el control de versiones*. https://www.atlassian.com/es/git/tutorials/what-is-version-control

Rodríguez, D. (2018, 21 octubre). *Introducción a los sistemas de control de versiones con Git*. Analytics Lane. https://www.analyticslane.com/2018/10/24/introduccion-a-los-sistemas-de-control-de-versiones-con-git/

Git - Acerca del Control de Versiones. (s. f.-b). .
https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

S. (2021, mayo 19). *¿Qué es el control de versiones? - Azure DevOps*. Microsoft Docs.
https://docs.microsoft.com/es-es/devops/develop/git/what-is-version-control

Docunecta. (2020, noviembre 26). *Control de versiones de documentos: sistema y software*.
https://www.docunecta.com/blog/control-de-versiones-de-documentos