****

****

**Articulo Técnico SVN**

**Subversion Basics**

**2016**

Introducción

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, **Subversion** maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Ésto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo”.

En este documento se detallaran las partes más importantes que recalcan la importancia y el enorme beneficio que esto trae al software.

Definición e historia

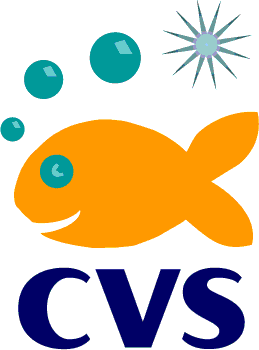
**Subversion** es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para los programadores, esos ficheros pueden ser código fuente, para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá.

En febrero del 2000, contactaron con Karl Fogel, autor de Open Source Development with **CVS** (Coriolis, 1999), y le preguntaron si le gustaría trabajar en este nuevo proyecto.

Casualmente, por aquel entonces Karl ya se encontraba discutiendo sobre el diseño de un nuevo sistema de control de versiones con su amigo Jim Blandy. En 1995, los dos habían fundado **Cyclic Software**, compañía que hacía contratos de soporte de **CVS**, y aunque después vendieron el negocio, seguían usando **CVS** todos los días en sus trabajos.



La frustración de ambos con **CVS** había conducido a Jim a pensar cuidadosamente acerca de mejores vías para administrar datos versionados , y no sólo tenía ya el nombre de “**Subversion**”, sino también el diseño básico del repositorio de **Subversion**.



Cuando **CollabNet** llamó, Karl aceptó inmediatamente trabajar en el proyecto, y Jim consiguió que su empresa, **RedHat Software**, básicamente lo donara al proyecto por un período de tiempo indefinido. **Collabnet** contrató a Karl y a Ben Collins-Sussman, y el trabajo detallado de diseño comenzó en mayo del 2000.

Subversion atrajo rápidamente a una comunidad activa de desarrolladores. Ésto vino a demostrar que era mucha la gente que había tenido las mismas frustrantes experiencias con **CVS**, y que había recibido con agrado la oportunidad de hacer algo al respecto.

El equipo de diseño original establece algunos objetivos simples. No querían abrir nuevos caminos en la metodología del control de versiones, sólo querían corregir **CVS**.Decidieron que **Subversion** incorporaría las características de **CVS**, y que preservarán el mismo modelo de desarrollo, pero sin duplicar los defectos obvios de **CVS**. Y aunque no necesitaba ser un reemplazo exacto de **CVS**, debía ser lo bastante similar para que cualquier usuario de **CVS** pudiera hacer el cambio con poco esfuerzo.

Después de catorce meses de codificación, **Subversion** pasó a ser “auto-hospedado” el 31 de agosto del 2001. Es decir, los desarrolladores de **Subversion** dejaron de usar **CVS** para la administración del propio código fuente de **Subversion**, y en su lugar empezaron a usar **Subversion**.

Si bien fue **CollabNet** quien inició el proyecto, y todavía financia una gran parte del trabajo (paga el salario de unos pocos desarrolladores a tiempo completo de **Subversion**), **Subversion** funciona como la mayoría de proyectos de código abierto, dirigido por un conjunto informal de reglas transparentes que fomentan el mérito. La licencia copyright de **CollabNet** es completamente compatible con las Directrices de Software Libre de Debian (**ver Referencias[3] *Directrices de software libre de Debian***).

En otras palabras, cualquier persona es libre de descargar, modificar, y redistribuir Subversion como desee; no se requiere ningún permiso de **CollabNet** o de cualquier otra persona.

Características

Versionado de directorios

Subversión implementa un sistema de ficheros versionado “virtual” que sigue los cambios sobre árboles de directorios completos a través del tiempo. CVS solamente lleva el historial de ficheros individuales.

Verdadero historial de versiones

Subversión puede añadir, borrar, copiar, y renombrar ficheros y directorios. Y cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo.

Envíos atómicos

Una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.

Elección de las capas de red

Subversion Cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor provee a la hora de autentificar archivos (SQL, LDAP, PAM). También tiene disponible un servidor de Subversion independiente, y más ligero.

Manipulación consistente de datos

Subversion maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).

Ramificación y etiquetado eficientes

Subversion crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman poco tiempo

Hackability

Subversion no tiene un equipaje histórico; está implementado como una colección de bibliotecas compartidas en C con APIs bien definidas. Ésto hace a Subversion extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes.

Herramientas que utilizan subversión

**TortoiseSVN:** TortoiseSVN es un cliente [Subversion](https://es.wikipedia.org/wiki/Subversion), implementado como una extensión al Shell de [Windows](https://es.wikipedia.org/wiki/Windows). Es [software libre](https://es.wikipedia.org/wiki/Software_libre) liberado bajo la licencia [GNU GPL](https://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_de_GNU).

Tortoise  administrador archivos y directorios a lo largo del tiempo, archivos se almacenan en un repositorio central y el repositorio es prácticamente lo mismo que un servidor de archivos con la excepción de que recuerda todos los cambios que se hayan hecho a sus archivos y directorios.

Características de tortoise

·         Integración con los comandos de Windows

·         Iconos superpuestos (sirven para verificar el estado en que se encuentra su copia de trabajo.

·         Interfaz Gráfica de Usuario

·         Fácil acceso a los comandos de Subversion

**RapidSVN: RapidSVN** es una multiplataforma [de Subversión](https://en.wikipedia.org/wiki/Subversion_(software)) cliente. Es liberado bajo la [Licencia Pública General de GNU](https://en.wikipedia.org/wiki/GNU_General_Public_License) .

Características

·         Simple - proporciona una interfaz fácil de usar para las características de Subversión

·         Eficiente - simple para los principiantes pero lo suficientemente flexible como para aumentar la productividad para los usuarios de Subversión con experiencia

·         Portable - se ejecuta en cualquier plataforma en la que Subversión y wxWidgetspuede ejecutar: Linux, Windows, Mac OS / X, Solaris, etc.

·         Rápido - completamente escrito en C + +

·         Multilingüe - que ha sido traducido a muchos idiomas ya: alemán, francés, italiano, portugués, ruso, ucraniano, chino simplificado, japonés

·         Soporte completo para Unicode.

**RabbitVCS:** [RabbitVCS](http://www.rabbitvcs.org/) es un conjunto de [herramientas gráficas](http://wiki.rabbitvcs.org/wiki/about/screenshots) implementadas para proporcionar un acceso simple y directo al sistema de control de versiones que utilizas. Actualmente, está integrado en Nautilus y solamente soporta Subversión. RabbitVCS está inspirado en [TortoiseSVN](http://tortoisesvn.tigris.org/)

Características

·         Integración con Nautilus.(Nautilus es un sistema operativo)

·         Diálogos de gran usabilidad.

·         Soporte para varios idiomas (**internacionalización**), actualmente soporta 16 idiomas.

·         Instalación fácil de acorde a tu distribución Linux (Ubuntu, Debian, Fedora, Gentoo, Arch Linux).

·         Integración con la línea de comandos.

Conversor de repositorio

**cvs2svn:** cvs2svn es una herramienta para la migración de un repositorio CVS(Sistema Concurrente de Versiones) a Subversion, Git, o bazar. Los principales objetivos de diseño son la robustez y el 100% preservación de datos.

* El objetivo principal de cvs2svn es migrar la mayor cantidad de información posible de su repositorio CVS antiguo, al nuevo Subversion o git repositorio.
* Por desgracia, CVS no registra información completa sobre la historia de su proyecto. Por ejemplo, CVS no registra qué modificaciones de archivos se llevó a cabo junto en los mismos cambios en el CVS. Por lo tanto, cvs2svn intenta deducir a partir de información incompleta de CVS lo que realmente sucedió en la historia de su repositorio. Así que el segundo objetivo de la cvs2svn es reconstruir tanto de la historia de su repositorio CVS como sea posible.
* El tercer objetivo de cvs2svn es que le permite personalizar el proceso de conversión y la forma de su repositorio de salida de la mayor flexibilidad posible. cvs2svn tiene muchas opciones de conversión que se pueden utilizar desde la línea de comandos, muchos más que se puede configurar a través de un archivo de opciones, y proporciona muchos ganchos para permitir la personalización aún más extrema escribiendo código Python.

Herramientas de mayor nivel

**Subissue:** Subissue tiene como objetivo eliminar la base de datos del software tradicional de gestión de fallos, utilizando [Subversion](http://subversion.tigris.org/) como su base. Es decir, en lugar de almacenar los problemas en una base de datos relacional, los problemas se almacenan directamente en el depósito de subversión.

Subissue gira en torno a tres grandes partes: la especificación de almacenamiento, la interfaz basada en la web, y un pre-entrega de la escritura para la integración directa de Subversion.

Ciclo básico de trabajo

Subversion cuenta con una gran cantidad de opciones para su uso, pero bajo uso cotidiano lo más probable es que use solamente unas de ellas.

1. **Actualizar la copia de trabajo**

A la hora de trabajar en un proyecto con un equipo, el paso más importante es actualizar la copia de trabajo local. Subversion incluye las modificaciones que se realizaron en los archivos desde la última vez que se actualizó por otros desarrolladores en el proyecto, es decir que pone la copia de trabajo en sincronía con la última revisión en el repositorio.

Cuando el servidor envía cambios a la copia de trabajo local, un código de letras es mostrado al lado de cada elemento para hacerle saber qué acciones realizó Subversion para actualizar la copia de trabajo local:

|  |  |
| --- | --- |
| **U** | El archivo fue actualizado (Updated). |
| **A** | El archivo o directorio fue Añadido a la copia de trabajo local. |
| **D** | El archivo o directorio fue borrado (Deleted) de la copia de trabajo local. |
| **R** | El archivo o directorio fue Reemplazado en la copia de trabajo local; es decir que fue borrado, y un nuevo archivo con el mismo nombre fue añadido. Aunque pueden tener el mismo nombre, el repositorio los considera objetos distintos con historiales distintos. |
| **G** | El archivo recibió nuevos cambios del repositorio, pero la copia local del archivo tenía las modificaciones que ya se había hecho. También puede que los cambios no se intersectaron que eran exactamente iguales que sus modificaciones locales, así que Subversion ha fusionado (merGed) satisfactoriamente los cambios del repositorio en el archivo sin ningún problema. |
| **C** | El archivo recibió cambios Conflicting del servidor. Los cambios del servidor directamente se añadieron sobre los cambios realizados en el archivo. Esta superposición necesita ser resuelta manualmente. |

1. **Hacer cambios en la copia de trabajo**

Cambios que puede hacer a su copia de trabajo local:

1. **Cambios en los archivos**

Subversion detecta automáticamente los cambios y los archivos a los que fueron aplicados dichos cambios.

1. **Cambios en el árbol**

Subversion ofrece la opción de marcar los archivos y directorios para el borrado planificado, la adición, copia o movimientos. Sin embargo, los cambios en el árbol solamente se llevarán a cabo hasta que el usuario envíe los mismos al servidor. A continuación se detallan los 4 comandos que hacen posible lo indicado:

|  |  |
| --- | --- |
| **add** | Programa añadir proyectos al repositorio. Cuando haga su próximo envío, el proyecto se convertirá en hijo de su directorio padre. Fíjese que si el proyecto es un directorio, todo por debajo del mismo será programado para la adición. Si solo quiere añadir el propio proyecto, pase la opción **--non-recursive** (**-N**). |
| **delete** | Programa borrar proyectos del repositorio. Si el proyecto es un archivo, se borrará inmediatamente de su copia de trabajo local. Si el proyecto es un directorio, este no es borrado, pero Subversion lo programa para borrarlo. Cuando envíe sus cambios, el proyecto será borrado de su copia de trabajo y del repositorio. Por supuesto, nada es borrado totalmente del repositorio, solo del HEAD del repositorio. |
| **copy** | Crea un nuevo objeto como duplicado del proyecto. La copia es automáticamente programada para la adición. Cuando la copia es añadida al repositorio en el siguiente envío de cambios, su historia de copia es registrada (como que originalmente viene del proyecto). |
| **move** | Funciona exactamente igual que **copy** y **delete**. Es decir, se programa la copia para la adición como una copia del proyecto, y se programa el proyecto para la eliminación. |

1. **Examinar los cambios**

Una vez hecho los cambios, estos necesitan ser enviados al repositorio, sin embargo, es recomendable revisar que se realizaron correctamente, teniendo la posibilidad de invertirlos antes de publicarlos.

Subversion ha sido optimizado para hacer muchas cosas sin comunicarse con el repositorio. En particular, la copia de trabajo local contiene una copia “antigua” secreta almacenada de cada archivo de versión controlado dentro del área **.svn**. Debido a esto, Subversion puede rápidamente mostrarle cómo sus archivos de trabajo han cambiado.

**svn status**

Este comando mostrará toda la información que se necesita saber sobre las modificaciones realizadas en la copia de trabajo local. Si se ejecuta en la raíz de la copia de trabajo, éste reconoce todos los cambios que se haya realizado en archivos.

A continuación se muestran los diferentes códigos de estado que **svn status** puede devolver:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| $ svn status | | | | | |
|  |  | L |  |  | abc.c |
| M |  |  |  |  | bar.c |
|  | M |  |  |  | baz.c |
| X |  |  |  |  | 3rd\_party |
| ? |  |  |  |  | foo.o |
| ! |  |  |  |  | some\_dir |
| ~ |  |  |  |  | Qux |
| I |  |  |  |  | .screenrc |
| A |  |  | + |  | moved\_dir |
| M |  |  | + |  | moved\_dir/README |
| D |  |  |  |  | stuff/fish.c |
| A |  |  |  |  | stuff/loot/bloo.h |
| C |  |  |  |  | stuff/loot/lump.c |
|  |  |  |  | S | stuff/squawk |

En este formato, **svn status** imprime cinco columnas:

**1° Columna**: indica el estado de un archivo o directorio y/o su contenido:

* **M:** el contenido del archivo ha sido modificado.
* **X:** el directorio está sin versionar.
* **?:** el archivo o directorio no está bajo control de versiones.
* **!:** el archivo o directorio está bajo el control de versiones pero falta o está incompleto. El comando **svn update** repondrá el archivo o el directorio desde el repositorio, o **svn revert** restaurará un archivo que falta.
* **~:** el archivo o directorio está en el repositorio como un tipo de objeto, pero actualmente está en su copia de trabajo como otro tipo.
* **I:** Subversion está “ignorando” el archivo o directorio, probablemente porque le fue indicado.
* **A:** el archivo o directorio ha sido programado para la adición en el repositorio.
* **D:** el archivo o directorio ha sido programado para la eliminación en el repositorio.
* **C:** el archivo está en un estado de conflicto. Es decir que los cambios recibidos del servidor durante una actualización se cubren con cambios locales que se tiene en la copia de trabajo.

**2° Columna**: indica el estado de las propiedades de un archivo o un directorio. Si aparece una M, entonces las propiedades han sido modificadas, sino, se imprime un espacio en blanco.

**3° Columna**: mostrará un espacio en blanco o una L, indicando que Subversion ha bloqueado el objeto en el área de trabajo **.svn**. Se verá una L si ejecuta **svn status** en un directorio donde un **svn commit** esté en progreso, quizás cuando esté editando el informe de cambios.

**4° Columna**: mostrará un espacio blanco o un +, indicando que el archivo o directorio está programado para ser añadido o modificado con historial adjunto. Esto ocurre típicamente cuando se utiliza **svn move** o **svn copy** enun archivo o directorio. El signo de + significa que el objeto es parte de un subárbol programado para la adición con historial. Cuando envíe los cambios, primero el padre será añadido con historial (copiado), lo que significa que este archivo existirá automáticamente en la copia. Entonces las modificaciones locales serán enviadas al repositorio.

**5° Columna**: mostrará un espacio en blanco o una S. Esto significa que el archivo o directorio ha sido movido de la ruta del resto de la copia de trabajo a una rama (**svn switch**).

También se puede especificar una ruta a la hora de ejecutar **svn status**, el cual le dará información acerca de ese objeto solamente.

El comando **svn status** también tiene una opción: **--verbose** (**-v**), el cuál le mostrará el estado de todos los objetos en su copia de trabajo, incluso si este no ha sido cambiado. La primera columna permanece igual, pero la segunda columna muestra la revisión de trabajo del archivo. La tercera y cuarta columna muestra la revisión en la cual el objeto cambió por última vez, y quién lo cambió.

Finalmente está la opción **--show-updates** (**-u**), la cual contacta con el repositorio y añade información acerca de las cosas que están fuera de fecha, los cuales son mostrados con un asterisco.

**svn diff**

Este comando puede descubrir exactamente cómo ha modificado archivos. Compara los archivos de la copia de trabajo contra la copia “antigua” almacenada en el área **.svn**.

La salida es visualizada en formato unificado del diff. Es decir, las líneas quitadas son empezadas con un - y las líneas añadidas son empezadas con un +.

**svn revert**

Este comando se utiliza cuando se desea invertir una modificación que se hizo de manera accidental a algún archivo, sustituyendo el mismo a un estado pre-modificado reescribiéndolo con la copia “antigua” almacenada en el área **.svn**. También puede deshacer cualquier operación programada, como la adición de un nuevo archivo o la supresión de un archivo del control de versión.

1. **Fusionar los cambios de otros en la copia de trabajo (Resolver conflictos)**

Siempre que aparezca un conflicto ocurre lo siguiente:

* Subversion imprime una C durante la actualización, y recuerda que el archivo está en un estado de conflicto.
* Subversion coloca marcas de conflicto, que son secuencias especiales de texto que delimitan los “lados” del conflicto en el archivo para demostrar visualmente las áreas cubiertas.
* Para cada archivo en conflicto, Subversion coloca 3 archivos extra en su copia de trabajo local:
* **filename.mine**: este es el archivo como existió en la copia de trabajo antes de que actualizara, el cual posee los últimos cambios.
* **filename.r\_oldrev\***: este es el archivo que era la revisión BASE antes de que actualizara la copia de trabajo. Es decir, el archivo que descargó antes de que hiciera la última edición.
* **filename.r\_newrev\***: este es el archivo que el cliente de Subversion recibió del servidor justo cuando actualizó la copia de trabajo. Este archivo corresponde con la revisión HEAD del repositorio.

**\***Aquí \_oldrev es el número de revisión del archivo en su directorio .svn y \_newrev es el número de revisión del HEAD del repositorio.

Cuando esto suceda, Subversion no permitirá enviar el archivo hasta que los tres archivos temporales sean borrados, lo cual necesitar hacer una de 3 cosas:

* **Fusionando conflictos a mano**: el archivo al que se desea resolver los conflictos muestran los signos menor-que (<) y mayor-que (>), que son marcas de conflictos, y no son parte de los datos en conflicto actuales. Además, muestra una línea de signos de igual (=) que divide los realizado por los usuarios. Generalmente estos signos no tienen que aparecer antes del próximo envío. El texto entre las dos primeras marcas está compuestas por los cambios que el primer usuario hizo, y entre el segundo y tercer conjunto de marcas de conflicto es el del otro usuario. Una vez llegado a un acuerdo, deberá editar el archivo y borrar las marcas de conflicto.
* **Copiando uno de los archivos temporales sobre el archivo de trabajo:** si se obtiene un conflicto y se decide en rechazar los cambios, simplemente copia uno de los archivos temporales creados por Subversion sobre el archivo en la copia de trabajo.
* **Ejecutar svn revert <nombre\_archivo>:** si se obtiene un conflicto, y al examinarlo se decide en rechazar sus cambios y empezar la edición de nuevo, simplemente invierta los cambios locales. En este caso no debe ejecutar **svn resolved**.

Una vez que se haya resuelto el conflicto, se debe ejecutar el comando **svn resolved** para indicarle a Subversion que se enviarán los cambios. Automáticamente se eliminarán los 3 archivos temporales y Subversion no considerará por más tiempo que el archivo está en estado de conflicto incluso si todavía tiene marcas de conflicto. No se debe indicar argumentos para éste comando.

1. **Enviar sus cambios**

El comando **svn commit** permite enviar todos los cambios al repositorio. Cuando se envía un cambio, necesita proveer un mensaje de registro, describiendo su cambio. El mensaje de registro será adjuntado a la nueva revisión que se ha creado. Si el mensaje de registro es breve, puede proveerlo en la línea de comando **--message** (o **-m**).

Sin embargo, si se crea un mensaje de registro mientras se trabaja, se le puede indicar a Subversion que tome el mensaje de un archivo, pasando el nombre del archivo con la opción **--file**.

Si se falla en especificar cualquiera de las opciones **--message** o **--file**, entonces Subversion lanzará automáticamente su editor para redactar un mensaje de registro.

Si se está en el editor escribiendo un mensaje de registro y se decide cancelar el envío, se puede quitar el editor sin guardar los cambios. Si ya se ha guardado el mensaje de registro, simplemente se borra el texto y se salva otra vez.

**Comparación del Ciclo de control de versiones contra Team Foundation**

Team Foundation (TFS) es un sistema de control de versiones que utiliza el modelo de repositorio de Cliente – Servidor al igual que Subversion (SVN). Pero ambos poseen características que las hacen diferentes en el ciclo de control. A continuación se mencionara las características relevantes en el ciclo de control de versiones:

Una vez integrado un proyecto al control de versión se crea una copia local la cual se pueden almacenar, editar o eliminar. En el caso de SVN como fue previamente explicado cuando en el repositorio envía los cambios a una copia local envía que cambio se realizó mediante un código de letras, en el caso de TFS los cambios realizados en el servidor (independiente de la acción), no especifica textualmente al cliente el cambio, simplemente realiza el cambio en el proyecto en el momento que descarga la última versión.

En el uso de sistema de control de versiones desde líneas de comando, TFS y SVN manejan diferentes comandos para realizar las diferentes tareas que realizan dichos sistema. En el caso de TFS cuando se utiliza con la herramienta de Visual Studio el uso de comandos es ignorado debido que utiliza Interfaz dentro de la herramienta.

Dentro del ciclo de control de Versiones de TFS, en la página de web de Microsoft divide en 2 las principales funciones, las cuales son: Colaborador (Utiliza normalmente el control de versiones para agregar, eliminar y modificar archivos con el fin de mantener un registro histórico de todas las modificaciones) y el Administrador (administra el servidor de control de versiones manteniendo la integridad de los datos almacenados en éste y se encarga de administrar los grupos de trabajo, los permisos, la instalación y las directivas de protección), con más detalle se especifica las acciones que realizan:

1. Colaborador:
   1. Desproteger archivos en su área de trabajo local.
   2. Proteger archivos modificados.
   3. Agregar nuevos archivos al servidor de control de versiones.
   4. Eliminar archivos del servidor de control de versiones.
   5. Comparar archivos.
   6. Combinar cambios entre los archivos.
2. Administrador:
   1. Configurar las opciones del entorno de control de versiones.
   2. Controlar el acceso a la base de datos de control de versiones.
   3. Destruir archivos o carpetas con control de versiones.

El ciclo común de trabajo de Subversion explicada en el aparto anterior muestra que trabaja similar que TFS por ejemplo se maneja de la siguiente manera: Actualizar la copia local, hacer cambios en los archivos, examinar los cambios realizados, revertir los cambios realizados, resolver los conflictos cuando se realiza el Merge con los cambios de otros y el commit de los cambios.

Conclusión

Los sistemas de control de versiones ofrecen un soporte muy importante en el proceso de desarrollo de software, colaborando en la gestión del control de versiones de los archivos de código fuente e impulsando el trabajo colaborativo al permitir que múltiples desarrolladores trabajen paralelamente en un mismo proyecto en forma autónoma e independiente.

Adicionalmente, los sistemas de control de versiones proporcionan los mecanismos para que cualquier versión de un proyecto pueda ser recuperada para visualizarse o modificarse, desplegando las diferencias entre las versiones existentes. Además se define el proceso de creación y actualización de repositorios, tanto en el enfoque centralizado como distribuido, detallándose el proceso de actualización de archivos de código fuente de acuerdo a las herramientas vistas.

Referencias

Ben Collins-Sussman, B. W. (n.d.). *Control de versiones con Subversion.*

Edgar Tello-Leal, C. M.-L. (2012). *REVISIÓN DE LOS SISTEMAS DE CONTROL DE VERSIONES UTILIZADOS.*