Control de versiones con Subversion

Revision 5586

Ben Collins-Sussman
Brian W. Fitzpatrick
C. Michael Pilato

Control de versiones con Subversion: Revision 5586

por Ben Collins-Sussman, Brian W. Fitzpatrick, y C. Michael Pilato

fecha de publicación (TBA)

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de ejemplos

Prólogo

Una mala lista de preguntas y respuestas frecuentes (FAQ[1].) es aquella compuesta a partir de las preguntas que el autor de la FAQ *desearía* que la gente hiciese, en lugar de las preguntas que la gente hace realmente. Quizás haya visto una de estas antes:

> P: ¿Cómo puedo usar Glorbosoft XZY para maximizar la productividad de mi equipo?
>
> R: Muchos de nuestros clientes desean saber cómo maximizar su productividad a través de nuestras innovaciones patentadas de software para trabajo en grupo. La respuesta es simple: primero, haga clic en el menú "`Archivo`", baje hasta "`Incrementar productividad`", entonces…

El problema con FAQs de este estilo es que no son, en un sentido literal, FAQs en absoluto. Nadie llamó nunca a la línea de soporte técnico y preguntó, "¿Cómo podemos maximizar la productividad?". En su lugar, la gente hizo preguntas altamente específicas, como, "¿Cómo podemos modificar el sistema de calendario para enviar recordatorios con dos días de antelación en lugar de uno?" y similares. Pero es mucho más fácil inventar preguntas y respuestas frecuentes que descubrirlas de verdad. Compilar una verdadera hoja de preguntas y respuestas frecuentes requiere una esfuerzo sostenido y organizado: durante el tiempo de vida del software las preguntas realizadas deben ser rastreadas, las respuestas monitorizadas, y todo ello recogido en un todo coherente e indexado que refleje la experiencia colectiva de los usuarios en ambientes de producción. Requiere de la actitud paciente y observadora de un naturalista de campo. Sin grandes hipótesis ni discursos visionarios—ojos abiertos y tomar notas con detalle es lo que más se necesita.

Lo que me encanta de este libro es que creció justo de un proceso similar, y se puede apreciar en cada página. Es el resultado directo de los encuentros que tuvieron los autores con los usuarios. Comenzó con la observación de Ben Collins-Sussman, que las mismas preguntas básicas estaban siendo realizadas una y otra vez en la lista de correo de Subversion: ¿Cuál es el flujo de trabajo estándar recomendado con Subversion? ¿Funcionan las etiquetas y ramas del mismo modo que en otros sistemas de control de versiones? ¿Cómo puedo averiguar quién ha hecho un cambio en particular?

Frustrado por ver las mismas preguntas día tras otro, Ben trabajó intensamente durante un mes en verano del 2002 para escribir *El Manual de Subversion* [2], un manual de sesenta páginas que cubría el uso básico de Subversion. El manual no tenía pretensiones de ser una obra completa, pero fue distribuido con Subversion y ayudó a los usuarios a superar ese bache inicial en su curva de aprendizaje. Cuando O'Reilly y Asociados decidió publicar un libro completo sobre Subversion, el camino del mínimo esfuerzo era obvio: expandir el manual de Subversion.

A los tres coautores del nuevo libro se les presentó una oportunidad sin igual. Oficialmente, su tarea consistía en escribir un libro de arriba hacia abajo, comenzando a partir de una tabla de contenidos y borrador inicial. Pero también tenían acceso a un flujo constante—de hecho, un géiser incontrolable—de material práctico. Subversion ya estaba en manos de miles de usuarios, y éstos estaban proporcionando toneladas de retroalimentación, no sólo sobre Subversion, sino también sobre la documentación existente.

Durante el tiempo que llevó escribir este libro, Ben, Mike y Brian frecuentaron las listas de correo y sesiones de chat de Subversion incesantemente, anotando cuidadosamente los problemas que estaban teniendo los usuarios en situaciones de la vida real. Monitorizar tal respuesta es de todos modos parte de la descripción de su puesto de trabajo en CollabNet, y les dio una gran ventaja cuando comenzaron a documentar Subversion. El libro producido está firmemente establecido sobre unos cimientos de experiencia, no en las arenas movedizas de las puras ilusiones; combina las mejores características de un manual de usuario y una lista de preguntas y respuestas frecuentes. Esta dualidad quizás no sea advertida durante una primera lectura. Leído en orden, de principio a fin, el libro es simplemente una descripción directa de una pieza de software. Hay una visión general, el obligado tour guiado, un capítulo sobre administración, algunos temas avanzados, y por supuesto una referencia de los comandos y una guía para resolver problemas. Sólo cuando se vuelve a revisar se ve brillar su autenticidad: los detalles que sólo pueden resultar de encuentros con lo inesperado, ejemplos extraídos de casos de uso genuinos, y sobre todo la sensibilidad hacia las necesidades del usuario y su punto de vista.

Por supuesto, nadie puede garantizar que este libro responderá cualquier pregunta que tenga sobre Subversion. A veces, la precisión con la que se anticipa a sus dudas le parecerá misteriosamente telepática; pero ocasionalmente, se topará con un agujero en el conocimiento de la comunidad de usuarios y se quedará con las manos vacías. Cuando esto ocurra, lo mejor que puede hacer es

[1] N.T.: El acrónimo proviene del inglés "Frequently Asked Questions".
[2] N.T.: The Subversion Handbook.

mandar un correo electrónico a <users@subversion.tigris.org> y exponer su problema. Los autores aun siguen ahí, observando, y esto no sólo incluye a los tres que aparecen en la portada, sino muchos otros que contribuyeron correcciones al material original. Desde el punto de vista de la comunidad de usuarios, solucionar su problema es meramente un agradable efecto colateral de un proyecto mucho mayor—en concreto, ajustar ligeramente este libro, y en definitiva Subversion, para que se adapte mejor a la forma en que la gente realmente lo usa. Están deseosos de escucharle, no únicamente porque le pueden ayudar, sino porque usted puede ayudarles a ellos. Con Subversion, como con todos los proyectos de software libre en activo, *usted no está solo*.

Deje que este libro sea su primer compañero.

— Karl Fogel, Chicago, 14 de Marzo, 2004

Prefacio

> “Si C le da suficiente cuerda para ahorcarse, piense en Subversion como una especie de almacén de cuerdas.”
> —Brian W. Fitzpatrick

En el mundo del software de código fuente abierto, Concurrent Versions System (CVS[1]) ha sido durante mucho tiempo la herramienta seleccionada para el control de versiones. Y con razón. CVS es software libre, y su *modus operandi* sin restricciones y soporte para trabajar en red—lo que permite a docenas de programadores dispersos geográficamente compartir su trabajo—se adapta muy bien a la naturaleza colaborativa del mundo de código fuente abierto. CVS y su modelo de desarrollo semi caótico se han convertido en la piedra angular de la cultura del código fuente abierto.

Pero al igual que muchas herramientas, CVS está haciéndose viejo. Subversion es un sistema de control de versiones relativamente nuevo diseñado para ser el sucesor de CVS. Los diseñadores se marcaron el objetivo de ganarse el corazón de los usuarios de CVS de dos modos: creando un sistema de código fuente abierto con un diseño (y “look and feel”) similar a CVS, e intentando corregir los defectos más notables de CVS. A pesar de que el resultado no representa necesariamente la siguiente gran evolución en diseño de los sistemas de control de versiones, Subversion *es* muy potente, muy fácil de usar y muy flexible.

Este libro documenta la serie 1.0 del sistema de control de versiones Subversion. Nos hemos esforzado por ser meticulosos. No obstante, Subversion cuenta con una próspera y energética comunidad de desarrolladores, por lo que, sin duda, ya habrá algunas características y mejoras planeadas para futuras versiones de Subversion que puedan modificar algunos comandos y notas específicas en este libro.

Audiencia

Este libro está orientado a los usuarios que saben manejar ordenadores y quieren usar Subversion para gestionar sus datos. Aunque Subversion puede ejecutarse en varios sistemas operativos diferentes, su interfaz de usuario principal es la línea de comandos. Es esta herramienta de línea de comandos (**svn**) la que será explicada y usada en este libro. Por consistencia, los ejemplos de este libro asumen que el lector está usando un sistema operativo tipo Unix y se siente relativamente cómodo con Unix y las interfaces de línea de comandos.

Dicho ésto, el programa **svn** también funciona en otras plataformas además de Unix como Microsoft Windows. A excepción de algunos detalles, como el uso de contra barras (\) en lugar de barras de dividir (/) como separadores en la ruta de los ficheros, los datos de entrada y de salida de esta herramienta cuando se ejecuta bajo Windows son idénticos a los de la versión para Unix. No obstante, es posible que los usuarios de Windows obtengan mejores resultados ejecutando los ejemplos bajo el entorno de emulación Unix Cygwin.

Probablemente, la mayoría de los lectores serán programadores o administradores de sistemas que necesiten seguir la pista a los cambios realizados sobre el código fuente. Éste es el uso más común de Subversion y, por lo tanto, el escenario que subyace en todos los ejemplos del libro. Pero Subversion puede usarse para administrar los cambios en cualquier tipo de información: imágenes, música, bases de datos, documentación, etc. Para Subversion, todos los datos no son más que datos.

A pesar de que este libro asume que el lector nunca ha usado un sistema de control de versiones, también hemos intentado facilitar a los usuarios de CVS realizar un cambio indoloro a Subversion. Pequeñas notas secundarias mencionarán CVS de vez en cuando, y un apéndice especial resume la mayoría de las diferencias entre CVS y Subversion.

Cómo leer este libro

Este libro intenta ser útil a personas con diferentes grados de experiencia—desde personas sin experiencia previa en el control de versiones hasta administradores de sistemas experimentados. Dependiendo de su propia experiencia, algunos capítulos le resultarán

[1] N.T.: Sistema concurrente de versiones. Debido a que el comando principal de este software se llama “cvs”, se usa la abreviación tanto para referirse al comando como al sistema completo y por ello no se traduce.

más o menos importantes. A continuación se detalla una “lista de lecturas recomendadas” para varios tipos de lectores:

Administradores de sistemas experimentados

Aquí asumimos que probablemente ya ha usado CVS antes, y que se muere por instalar y ejecutar un servidor Subversion lo antes posible. Los capítulos 5 y 6 le mostrarán cómo crear su primer repositorio y ofrecer acceso al mismo por red. Después de realizar estas tareas, el capítulo 3 y el apéndice A son las rutas más rápidas para aprender a manejar el programa cliente de Subversion aprovechando su experiencia con CVS.

Nuevos usuarios

Su administrador probablemente ya ha configurado el servidor de Subversion, así que necesita aprender a usar el cliente. Si nunca ha usado un sistema de control de versiones (como CVS), entonces los capítulos 2 y 3 son una introducción vital. Si ya tiene experiencia con CVS, el capítulo 3 y el apéndice A son los mejores sitios para comenzar.

Usuarios avanzados

Independientemente de que sea usuario o administrador, su proyecto acabará por crecer. Le interesará aprender cómo hacer cosas más avanzadas con Subversion, como por ejemplo usar ramas de desarrollo y realizar mezclas (capítulo 4), cómo usar la característica de propiedades de Subversion, cómo configurar las opciones de tiempo de ejecución (capítulo 7), y otras cosas. Los capítulos 4 y 7 no son vitales al principio, pero no olvide leerlos una vez se haya familiarizado con los conceptos básicos.

Desarrolladores

Asumimos que ya está familiarizado con Subversion y que ahora desea extender su funcionalidad o crear nuevo software usando sus múltiples APIs. El capítulo 8 está dedicado a usted.

El libro termina con material de referencia—el capítulo 9 es una guía de referencia de todos los comandos de Subversion, y los apéndices cubren varios temas interesantes. Una vez haya terminado de leer este libro, estos capítulos serán posiblemente los que vuelva a usar.

Convenciones empleadas en este libro

Esta sección cubre las convenciones empleadas en este libro.

Convenciones tipográficas

Anchura constante

Usada para comandos, salida de comandos y parámetros

`Anchura constante en cursiva`

Usada para elementos que se pueden reemplazar tanto en código fuente como texto

`Cursiva`

Usada para nombres de ficheros y directorios

Iconos

Este icono señala una nota relacionada con el texto al que acompaña.

Este icono señala una sugerencia útil relacionada con el texto al que acompaña.

Este icono señala un aviso relacionado con el texto al que acompaña.

Tenga en cuenta que los ejemplos de código fuente no son más que eso—ejemplos. Aunque puedan ser compilados con los comandos apropiados, su objetivo es el de ilustrar el problema expuesto, y no tienen por qué ser ejemplos de un buen estilo de programación.

Organización de este libro

Aquí tiene un listado de los siguientes capítulos y sus contenidos:

Capítulo 1, *Introducción*

Cubre la historia de Subversion, sus características, arquitectura, componentes y métodos de instalación. También incluye una guía rápida para comenzar.

Capítulo 2, *Conceptos básicos*

Explica los conceptos básicos del control de versiones y los diferentes modelos de versionado, así como el repositorio de Subversion, las copias de trabajo y las revisiones.

Capítulo 3, *Recorrido guiado*

Da un repaso a un día cualquiera en la vida de un usuario de Subversion. Muestra cómo usar Subversion para obtener, modificar y enviar cambios al repositorio.

Capítulo 4, *Crear ramas y fusionar cambios*

Explica las ramas, fusiones y etiquetado, incluyendo los mejores métodos para crear ramas y fusionarlas, casos de uso comunes, cómo deshacer cambios y cómo alternar fácilmente entre ramas de desarrollo.

Capítulo 5, *Administración de repositorios*

Describe los elementos básicos de un repositorio Subversion, cómo crear, configurar y mantener un repositorio, y las herramientas que puede usar para hacer todo esto.

Capítulo 6, *Configuración del servidor*

Explica cómo configurar su servidor Subversion y tres métodos de acceso: `HTTP`, el protocolo `svn` y el acceso local. También cubre los detalles de autenticación, autorización y acceso anónimo.

Capítulo 7, *Temas avanzados*

Explora los ficheros de configuración del cliente de Subversion, las propiedades de ficheros y directorios, cómo `ignorar` ficheros en su copia local, cómo incluir árboles externos en su copia local, y finalmente, cómo manejar ramas de desarrollo.

Capítulo 8, *Información para desarrolladores*

Describe detalles internos de Subversion, el sistema de ficheros de Subversion, y las áreas administrativas de una copia local

desde el punto de vista de un programador. Muestra cómo usar las APIs públicas para escribir un programa que use Subversion, y sobre todo, cómo contribuir al desarrollo de Subversion.

Capítulo 9, *Referencia completa de Subversion*

Explica con detalle todo subcomando de **svn**, **svnadmin**, y **svnlook** junto con una gran cantidad de ejemplos para todos los gustos.

Apéndice A, *Subversion para usuarios de CVS*

Cubre las similitudes y diferencias entre Subversion y CVS, con numerosas sugerencias para deshacerse de los malos hábitos adquiridos tras años de uso de CVS. Incluye descripciones de los números de revisión de Subversion, versionado de directorios, operaciones offline, **update** vs. **status**, ramas, etiquetas, metadatos, resolución de conflictos y autenticación.

Apéndice B, *Solución de problemas*

Dedicado a las dificultades y problemas habituales usando y compilando Subversion.

Apéndice C, *WebDAV y el auto versionado*

Describe los detalles de WebDAV y DeltaV, y cómo configurar su repositorio Subversion para que pueda ser montado como una recurso DAV compartido en modo lectura/escritura.

Apéndice D, *Herramientas de terceros*

Trata las herramientas que soportan o usan Subversion, incluyendo programas cliente alternativos, navegadores de repositorio y similares.

Este libro es libre

Este libro comenzó con fragmentos de documentación escritos por los desarrolladores del proyecto Subversion, los cuales fueron reescritos y agrupados en una única obra. Como tal, siempre ha estado bajo una licencia libre. (Lea Apéndice F, *Copyright*.) De hecho, el libro fue escrito bajo escrutinio público como parte de Subversion. Ésto significa dos cosas:

- Siempre encontrará la última versión de este libro en el propio árbol de código fuente de Subversion.
- Puede distribuir y realizar cambios en el libro a su gusto—está bajo una licencia libre. Por supuesto, en lugar de distribuir su propia versión privada de este libro, preferiríamos que enviase cualquier comentario o parche a la comunidad de desarrolladores Subversion. Lea "Contribuyendo a Subversion" para aprender cómo tomar parte en la comunidad.

Puede enviar comentarios y preguntas sobre la publicación a O'Reilly aquí: ###insert boilerplate.

Una versión online reciente de este libro puede encontrarse en `http://svnbook.red-bean.com`.

Agradecimientos

Este libro no hubiese sido posible (y no muy útil) si Subversion no existiese. Por esta razón, los autores quieren agradecer a Brian Behlendorf de CollabNet por su visión y apoyo económico a tan peligroso y ambicioso nuevo proyecto de código fuente abierto; Jim Blandy por el nombre original Subversion y diseño; te queremos, Jim; Karl Fogel por ser tan buen amigo y excelente líder de la comunidad, en ese orden.[2]

[2] Oh, y gracias Karl, por tener demasiado trabajo como para haber escrito este libro.

Gracias a O'Reilly y nuestros editores, Linda Mui y Tatiana Diaz por su paciencia y apoyo.

Finalmente, queremos dar gracias a las innumerables personas que contribuyeron al libro con revisiones informales, sugerencias y correcciones: A pesar de ser sin lugar a dudas una lista incompleta, este libro sería incompleto e incorrecto sin la ayuda de Jani Averbach, Ryan Barrett, Francois Beausoleil, Jennifer Bevan, Matt Blais, Zack Brown, Martin Buchholz, Brane Cibej, John R. Daily, Peter Davis, Olivier Davy, Robert P. J. Day, Mo DeJong, Brian Denny, Joe Drew, Nick Duffek, Ben Elliston, Justin Erenkrantz, Shlomi Fish, Julian Foad, Chris Foote, Martin Furter, Dave Gilbert, Eric Gillespie, Matthew Gregan, Art Haas, Greg Hudson, Alexis Huxley, Jens B. Jorgensen, Tez Kamihira, David Kimdon, Mark Benedetto King, Andreas J. Koenig, Nuutti Kotivuori, Matt Kraai, Scott Lamb, Vincent Lefevre, Morten Ludvigsen, Paul Lussier, Bruce A. Mah, Philip Martin, Feliciano Matias, Patrick Mayweg, Gareth McCaughan, Jon Middleton, Tim Moloney, Mats Nilsson, Joe Orton, Amy Lyn Pilato, Kevin Pilch-Bisson, Dmitriy Popkov, Michael Price, Mark Proctor, Steffen Prohaska, Daniel Rall, Tobias Ringstrom, Garrett Rooney, Joel Rosdahl, Christian Sauer, Larry Shatzer, Russell Steicke, Sander Striker, Erik Sjoelund, Johan Sundstroem, John Szakmeister, Mason Thomas, Eric Wadsworth, Colin Watson, Alex Waugh, Chad Whitacre, Josef Wolf, Blair Zajac, y toda la comunidad de Subversion.

De parte de Ben Collins-Sussman

Gracias a mi mujer Frances, quien, tras muchos meses llegó a oír, “Pero cariño, todavía estoy trabajando en el libro”, en lugar del habitual, “Pero cariño, todavía estoy escribiendo emails.” ¡No tengo ni idea de dónde saca toda su paciencia! Ella es mi contrapeso ideal.

Gracias a mi familia adquirida por su sincero apoyo, a pesar de no tener interés real en el tema. (Ya sabe, aquellos que cuando dicen, “Oh, ¿estás escribiendo un libro?”, y les respondes que es sobre ordenadores, desvían su mirada.)

Gracias a todos mis amigos íntimos, quienes enriquecen mi vida. No me miréis de ese modo—sabéis quienes sois.

De parte de Brian W. Fitzpatrick

Muchísimas gracias a mi mujer Marie por ser increíblemente comprensiva, apoyarme, y sobre todo, ser paciente. Gracias a mi hermano Eric quien me introdujo a la programación UNIX hace tiempo. Gracias a mi madre y abuela por todo su apoyo, sin olvidar aquellas vacaciones de navidades en las que tuvieron que soportarme porque nada más llegar a casa me dediqué a mi portátil para trabajar en el libro.

A Mike y Ben: ha sido un placer trabajar con vosotros en el libro. Vaya, ¡es un placer trabajar con vosotros en el trabajo!

A todas las personas de la comunidad Subversion y la Apache Software Foundation, gracias por aceptarme. No pasa un solo día sin que aprenda algo de al menos uno de vosotros.

Por último, gracias a mi abuelo, que siempre me decía “la libertad es responsabilidad.” No podría estar más de acuerdo.

De parte de C. Michael Pilato

Gracias en especial a mi mujer, Amy, por su amor y paciente apoyo, por soportar largas noches, e incluso por revisar secciones enteras de este libro—siempre das lo mejor, y lo haces con increíble elegancia. Gavin, cuando seas suficientemente mayor para leer, espero que estés tan orgulloso de tu papá como él lo está de ti. A mamá y papá (y el resto de la familia), gracias por vuestro constante apoyo y entusiasmo.

Me quito el sombrero ante Shep Kendall, a través de quien descubrí el mundo de los ordenadores; Ben Collins-Sussman, mi guía turístico por el mundo del código fuente abierto; Karl Fogel—tu *eres* mi `.emacs`; Greg Stein, por transmitirme conocimientos prácticos de programación. Brian Fitzpatrick—por compartir esta experiencia literaria conmigo. A todas las numerosas personas de las que siempre estoy aprendiendo cosas nuevas—¡continuad enseñando!

Por último, a Aquél que demuestra perfectamente excelencia creativa—gracias.

Capítulo 1. Introducción

El control de versiones es el arte del manejo de los cambios en la información. Ha sido durante mucho tiempo una herramienta crítica para los programadores, quienes normalmente empleaban su tiempo haciendo pequeños cambios en el software y después deshaciendo esos cambios al día siguiente. Pero la utilidad del software de control de versiones se extiende más allá de los límites del mundo del desarrollo de software. Allá donde pueda encontrarse a gente usando ordenadores para manejar información que cambia a menudo, hay un hueco para el control de versiones. Y aquí es donde entra en juego Subversion.

Este capítulo contiene una introducción general a Subversion — qué es; qué hace; cómo conseguirlo.

¿Qué es Subversion?

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 ”.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar mas rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados específicamente para la administración de árboles de código fuente, y tienen muchas características que son específicas del desarrollo de software— tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software. Sin embargo, Subversion no es uno de estos sistemas. Subversion es un sistema general que puede ser usado para administrar *cualquier* conjunto de ficheros. Para usted, 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.

Historia de Subversion

A principios del 2000, CollabNet, Inc. (`http://www.collab.net`) comenzó a buscar desarrolladores para escribir un sustituto para CVS. CollabNet ofrece un conjunto de herramientas de software colaborativo llamado SourceCast, del cual un componente es el control de versiones. Aunque SourceCast usaba CVS como su sistema de control de versiones inicial, las limitaciones de CVS se hicieron evidentes desde el principio, y CollabNet sabía que tendría que encontrar algo mejor. Desafortunadamente, CVS se había convertido en el estándar *de facto* en el mundo del código abierto porque *no había* nada mejor, al menos no bajo una licencia libre. Así CollabNet decidió escribir un nuevo sistema de control de versiones desde cero, manteniendo las ideas básicas de CVS, pero sin sus fallos y defectos.

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. Con la ayuda de algunos ajustes bien colocados de Brian Behlendorf y Jason Robbins de CollabNet, y Greg Stein (por aquel entonces un activo desarrollador independiente del proceso de especificación de WebDAV/DeltaV), 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 estableció 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 preservarían 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. 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 de Subversion

Al discutir acerca de las características que Subversion aporta al mundo del control de versiones, a menudo es útil hablar de ellas en términos de cómo han mejorado sobre el diseño de CVS. Si no está familiarizado con CVS, quizás no entienda todas estas características. Y si no está familiarizado con el control de versiones en absoluto, se le pueden nublar los ojos a menos que lea primero Capítulo 2, *Conceptos básicos*, donde proporcionamos una leve introducción al control de versiones en general.

Subversion proporciona:

Versionado de directorios

> CVS solamente lleva el historial de ficheros individuales, pero Subversion implementa un sistema de ficheros versionado “virtual ” que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.

Verdadero historial de versiones

> Dado que CVS está limitado al versionado de ficheros, operaciones como copiar y renombrar—las cuales pueden ocurrir sobre ficheros, pero que realmente son cambios al contenido del directorio en el que se encuentran—no son soportadas por CVS. Adicionalmente, en CVS no puede reemplazar un fichero versionado con algo nuevo que lleve el mismo nombre sin que el nuevo elemento herede el historial del fichero antiguo—que quizás sea completamente distinto al anterior. Con Subversion, usted 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. Ésto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.

Versionado de metadatos

> Cada fichero y directorio tiene un conjunto de propiedades —claves y sus valores —asociado a él. Usted puede crear y almacenar cualquier par arbitrario de clave/valor que desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

Elección de las capas de red

> Subversion tiene una noción abstracta del acceso al repositorio, facilitando a las personas implementar nuevos mecanismos de red. Subversion puede conectarse al servidor HTTP Apache como un módulo de extensión. Ésto proporciona a Subversion una gran ventaja en estabilidad e interoperabilidad, y acceso instantáneo a las características existentes que ofrece este servi-

dor—autenticación, autorización, compresión de la conexión, etcétera. También tiene disponible un servidor de Subversion independiente, y más ligero. Este servidor habla un protocolo propio, el cual puede ser encaminado fácilmente a través de un túnel SSH.

Manipulación consistente de datos

Subversion expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (ilegibles para humanos). Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red.

Ramificación y etiquetado eficientes

El coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Subversion crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante.

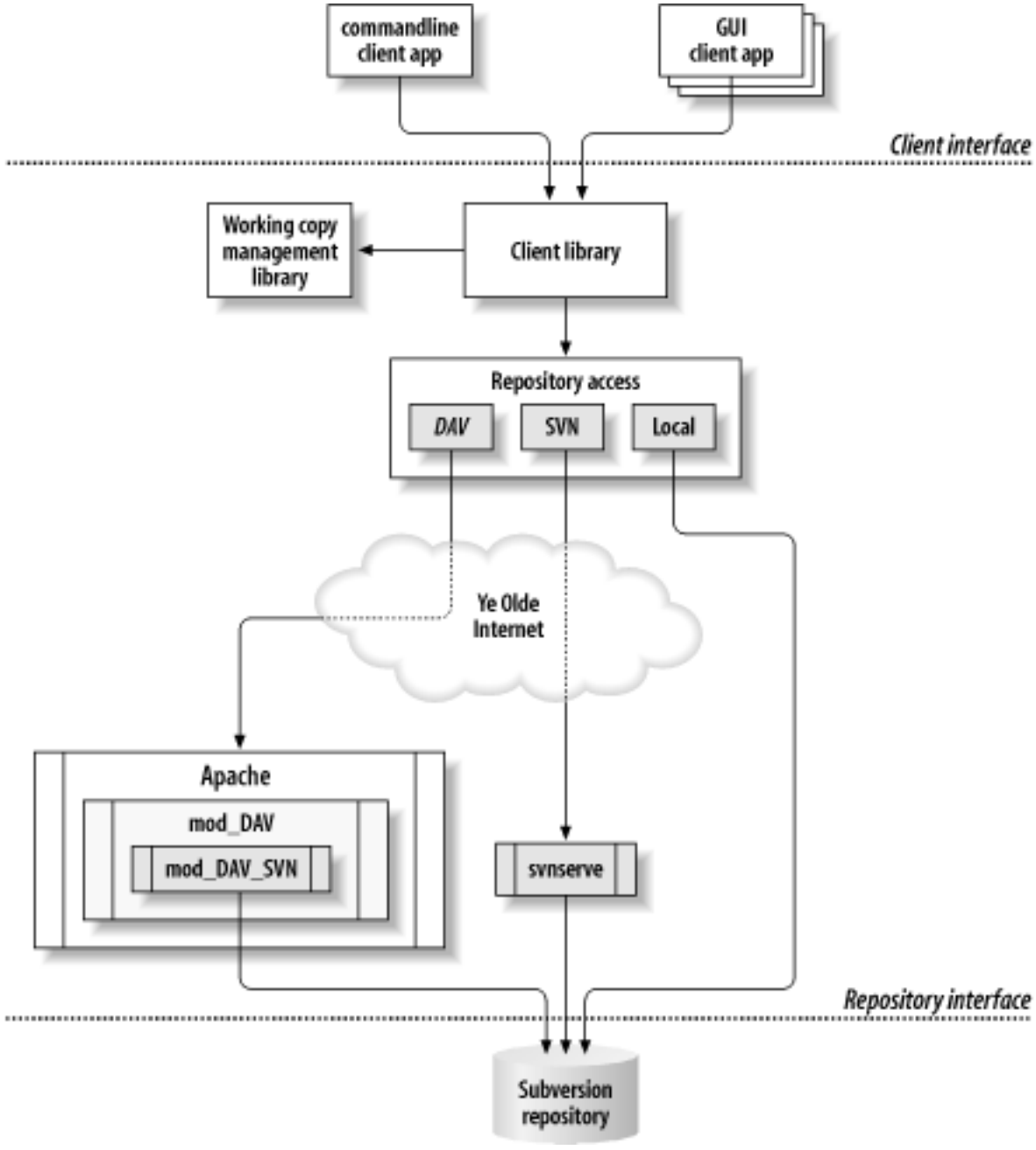
Hackability

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.

Arquitectura de Subversion

Figura 1.1, “Arquitectura de Subversion” ilustra lo que uno podría titular una visión panorámica del diseño de Subversion.

Figura 1.1. Arquitectura de Subversion



En un extremo se encuentra un repositorio de Subversion que conserva todos los datos versionados. Al otro lado, hay un programa cliente Subversion que administra réplicas parciales de esos datos versionados (llamadas “copias de trabajo”). Entre estos extremos hay múltiples rutas a través de varias capas de acceso al repositorio (AR). Algunas de estas rutas incluyen redes de ordenadores y servidores de red que después acceden al repositorio. Otras pasan por alto la red y acceden al repositorio directamente.

Instalando Subversion

Subversion está construido sobre una capa de portabilidad llamada APR (la biblioteca Apache Portable Runtime), lo cual significa que Subversion debería funcionar en cualquier sistema operativo donde lo haga el servidor httpd Apache: Windows, Linux, todos los sabores de BSD, Mac OS X, Netware y otros.

La manera más sencilla de obtener Subversion es descargando un paquete binario construido para su sistema operativo. El sitio web de Subversion (`http://subversion.tigris.org`) dispone a menudo de estos paquetes disponibles para su descarga, publicados por voluntarios. El sitio web contiene generalmente paquetes que incluyen instaladores gráficos para los usuarios de los sistemas operativos de Microsoft. Si usted usa un sistema operativo Unix o similar, puede usar el sistema nativo de distribución de paquetes de su sistema (RPMs, DEBs, el árbol de ports, etc.) para obtener Subversion.

Alternativamente, usted puede compilar Subversion directamente a partir del código fuente. Del sitio web de Subversion, descargue la ultima versión del código fuente. Después de desempaquetarlo, siga las instrucciones del fichero `INSTALL` para compilarlo. Observe que cada paquete de código fuente que se publica contiene todo lo necesario para construir un cliente de línea de comandos capaz de comunicarse con un repositorio remoto (en particular, las bibliotecas apr, apr-util y neon). Sin embargo, las partes opcionales de Subversion tienen otras muchas dependencias, tales como la base de datos Berkeley DB y posiblemente el servidor web Apache. Si usted quiere hacer una compilación completa, asegúrese de tener todos los paquetes documentados en el fichero `INSTALL`. Si planea trabajar en el propio Subversion, puede usar su programa cliente para obtener la última y más reciente versión del código fuente. Este procedimiento está documentado en “Obtenga el código fuente”.

Componentes de Subversion

Una vez instalado, Subversion se compone de un número diferente de piezas. A continuación se presenta una visión general de estos componentes. No se alarme si las descripciones breves no le dejan las cosas muy claras —hay páginas *de sobra* en este libro dedicadas a aliviarle esa confusión.

svn

: El programa cliente de línea de comandos.

svnversion

: Programa para informar del estado (en términos de revisiones de los elementos presentes) de una copia de trabajo.

svnlook

: Una herramienta para inspeccionar un repositorio de Subversion.

svnadmin

: Herramienta para crear, modificar o reparar un repositorio de Subversion.

svndumpfilter

: Un programa para filtrar el formato de salida de volcado de repositorios Subversion.

mod_dav_svn

: Un módulo para el servidor HTTP Apache usado para hacer que su repositorio esté disponible a otros a través de una red.

svnserve

: Un servidor independiente, ejecutable como proceso demonio o invocable por SSH; otra manera de hacer que su repositorio esté disponible para otros a través de una red.

Suponiendo que ha instalado Subversion correctamente, debería estar preparado para comenzar. Los próximos dos capítulos le guiarán a través del uso de **svn**, el programa cliente de Subversion de línea de comandos.

Un comienzo rápido

Algunas personas tienen problemas para absorber una nueva tecnología leyendo un enfoque del tipo "arriba a abajo" como el que ofrece este libro. Esta sección es una introducción muy breve a Subversion, y está pensada para dar a los principiantes algo con lo que defenderse. Si usted es de los que prefiere aprender experimentando, la siguiente demostración le pondrá en marcha. A lo largo del camino, le iremos dando enlaces a los capítulos relevantes de este libro.

Si a usted le resulta completamente nuevo el concepto de control de versiones o el modelo “copiar-modificar-mezclar” usado tanto por CVS como por Subversion, debería leer Capítulo 2, *Conceptos básicos* antes de seguir adelante.

El siguiente ejemplo asume que usted tiene preparados tanto el cliente de línea de comandos de Subversion **svn**, como la herramienta administrativa **svnadmin**. También asume que su cliente **svn** ha sido compilado con soporte para la base de datos Berkeley DB. Puede comprobarlo ejecutando **svn --version** y asegurándose de que el modulo `ra_local` está disponible. Sin este módulo, el cliente no podrá acceder a URLs del tipo `file://`

Subversion almacena todos los datos versionados en un repositorio central. Para comenzar, cree un nuevo repositorio:

```
$ svnadmin create /path/to/repos
$ ls /path/to/repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

Este comando crea un nuevo directorio `/path/to/repos` que contiene un repositorio de Subversion. Asegúrese de que este directorio reside en un disco local y *no* compartido en red. Este nuevo directorio contiene principalmente una colección de ficheros de la base de datos Berkeley DB. Para más información sobre la creación y mantenimiento de repositorios, vea Capítulo 5, *Administración del Repositorio*.

A continuación, cree un árbol de ficheros y directorios para importar dentro del repositorio. Por razones que se aclararán más tarde (vea Capítulo 4, *Crear ramas y fusionarlas*), su estructura debería tener tres directorios en el primer nivel de la jerarquía llamados `branches`,`tags`, y `trunk`:

```
/tmp/project/branches/
/tmp/project/tags/
/tmp/project/trunk/
               foo.c
               bar.c
               Makefile
               …
```

Una vez tenga un árbol de datos listo para continuar, impórtelo dentro del repositorio con el comando **svn import** (vea “**svn import**”):

```
$ svn import /tmp/project file:///path/to/repos -m "initial import"
Adding         /tmp/project/branches
Adding         /tmp/project/tags
Adding         /tmp/project/trunk
Adding         /tmp/project/trunk/foo.c
Adding         /tmp/project/trunk/bar.c
Adding         /tmp/project/trunk/Makefile
…
Committed revision 1.
$
```

Ahora el repositorio contiene este árbol de datos. Observe que el directorio original `/tmp/project` no se ha modificado; Subversion no se preocupa por él (de hecho, puede incluso borrar ese directorio si lo desea). Para comenzar a manipular los datos del repositorio, necesitará crear una nueva “copia de trabajo” de los datos, una especie de entorno de trabajo privado. Pida a Subver-

sion que “obtenga” [1] una copia de trabajo del directorio `trunk` del repositorio:

```
$ svn checkout file:///path/to/repos/trunk project
A  project/foo.c
A  project/bar.c
A  project/Makefile
…
Checked out revision 1.
```

Ahora usted dispone de una copia personal de parte del repositorio en un nuevo directorio llamado `project`. Puede editar los ficheros en su copia de trabajo y después depositar esos cambios de nuevo en el repositorio.

- Entre en su copia de trabajo y edite el contenido de un fichero.
- Ejecute **svn diff** para ver las diferencias introducidas por sus cambios en formato diff unificado.
- Ejecute **svn commit** para depositar la nueva versión de su fichero en el repositorio.
- Ejecute **svn update** para “sincronizar” su copia de trabajo con el repositorio.

Para un recorrido completo por todas las operaciones que puede realizar con su copia de trabajo, vea Capítulo 3, *Recorrido guiado*.

Llegado este punto, usted tiene la opción de hacer que su repositorio Subversion esté disponible a otros a través de una red. Vea Capítulo 6, *Configuración del servidor* para aprender acerca de los diferentes tipos de procesos servidor disponibles y cómo configurarlos.

[1] N.T.: En la bibliografía sobre control de versiones se suele utilizar el vocablo inglés «check out» para referirse a la operación usada para obtener una copia (parcial) de un repositorio centralizado. En ocasiones, la obtención de dicha copia implica la conexión a un servidor remoto, por lo que en la traducción es común emplear indistintamente los términos «obtener» y «descargar» para referirse a esta operación.

Capítulo 2. Conceptos básicos

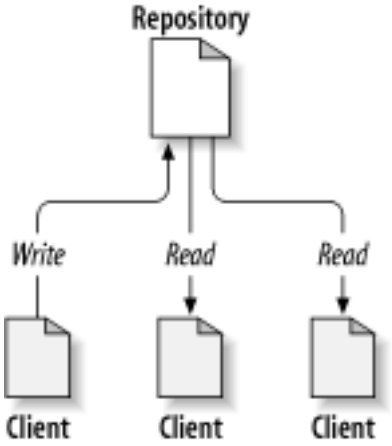
Este capítulo es una introducción breve e informal a Subversion. Si es nuevo en el tema del control de versiones, este capítulo es definitivamente para usted. Empezaremos tratando los conceptos generales en el control de versiones, seguiremos con las ideas específicas detrás de Subversion, y mostraremos algunos ejemplos simples de Subversion en acción.

Aunque los ejemplos de este capítulo muestran a gente compartiendo colecciones de archivos de código fuente, tenga en mente que Subversion puede manejar cualquier tipo de colección de archivos—no está limitado a asistir a programadores de ordenadores.

El repositorio

Subversion es un sistema centralizado para compartir información. La parte principal de Subversion es el repositorio, el cual es un almacén central de datos. El repositorio guarda información en forma de *árbol de archivos*—una típica jerarquía de archivos y directorios. Cualquier número de *clientes* puede conectarse al repositorio y luego leer o escribir en esos archivos. Al escribir datos, un cliente pone a disposición de otros la información; al leer datos, el cliente recibe información de otros. La figura Figura 2.1, “Un sistema cliente/servidor típico” ilustra ésto.

Figura 2.1. Un sistema cliente/servidor típico



Entonces, ¿qué tiene ésto de interesante?. Hasta ahora, suena como la definición del típico servidor de archivos. Y, de hecho, el repositorio *es* una especie de servidor de archivos, pero no del tipo habitual. Lo que hace especial al repositorio de Subversion es que *recuerda todos los cambios* hechos sobre él: cada cambio a cada archivo, e inclusive cambios al propio árbol de directorios, tales como la adición, borrado y reubicación de archivos y directorios.

Cuando un cliente lee datos del repositorio, normalmente sólo ve la ultima versión del árbol de archivos. Sin embargo, el cliente también tiene la posibilidad de ver estados *previos* del sistema de archivos. Por ejemplo, un cliente puede hacer consultas históricas como, “¿Qué contenía este directorio el miércoles pasado?” Esta es la clase de preguntas que resulta esencial en cualquier *sistema de control de versiones*: sistemas que están diseñados para registrar y seguir los cambios en los datos a través del tiempo.

Modelos de versionado

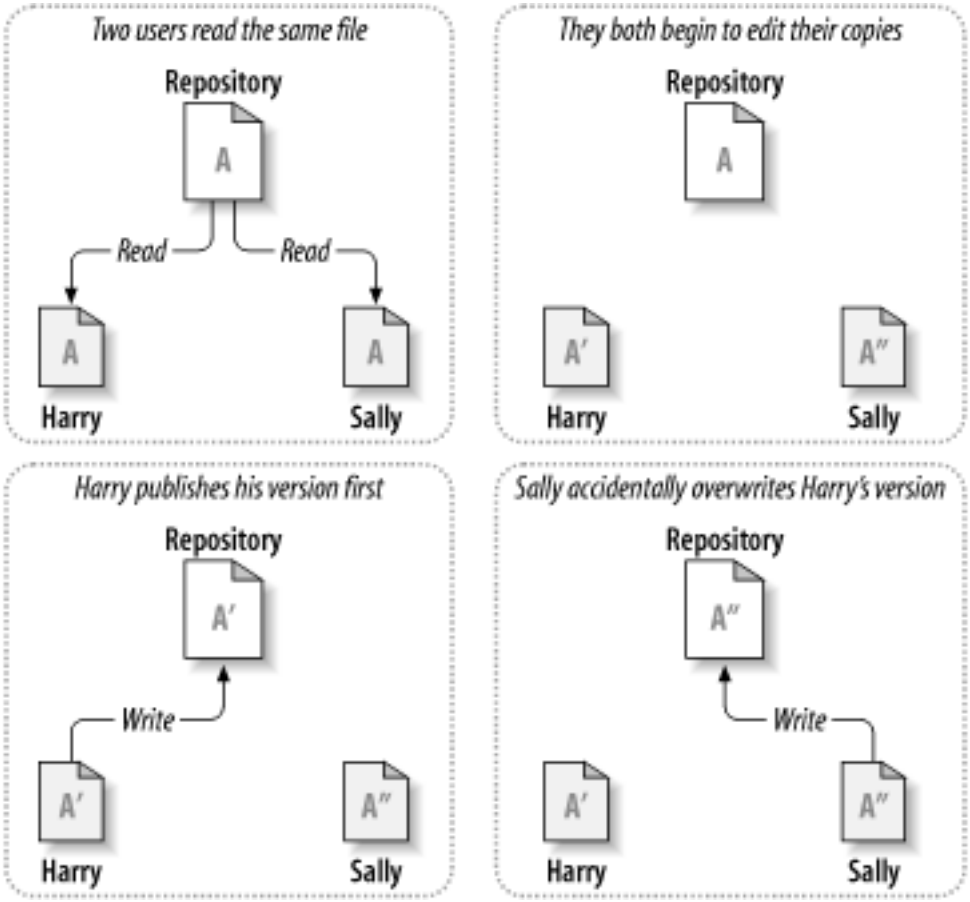
La misión principal de un sistema de control de versiones es permitir la edición colaborativa y la compartición de los datos. Sin embargo, existen diferentes sistemas que utilizan diferentes estrategias para alcanzar este objetivo.

El problema de compartir archivos

Todos los sistemas de control de versiones tienen que resolver un problema fundamental: ¿Cómo permitirá el sistema a los usuarios el compartir información, pero al mismo tiempo impedirá que se pisen los callos mutuamente de forma accidental? Es muy sencillo para los usuarios el sobreescribir accidentalmente los cambios de los demás en el repositorio.

Considere el escenario mostrado en Figura 2.2, “El problema a evitar”. Suponga que tenemos dos colaboradores, Juan y Carmen. Cada uno de ellos decide editar el mismo archivo del repositorio al mismo tiempo. Si Juan guarda sus cambios en el repositorio en primer lugar, es posible que (unos momentos más tarde) Carmen los sobreescriba accidentalmente con su propia versión del archivo. Si bien es cierto que la versión de Juan no se ha perdido para siempre (porque el sistema recuerda cada cambio), cualquier cambio que Juan haya hecho *no* estará presente en la versión más reciente de Carmen porque, para empezar, ella nunca vio los cambios de Juan. El trabajo de Juan sigue efectivamente perdido—o al menos ausente en la última versión del archivo—y probablemente por accidente. ¡Esta es definitivamente una situación que queremos evitar!

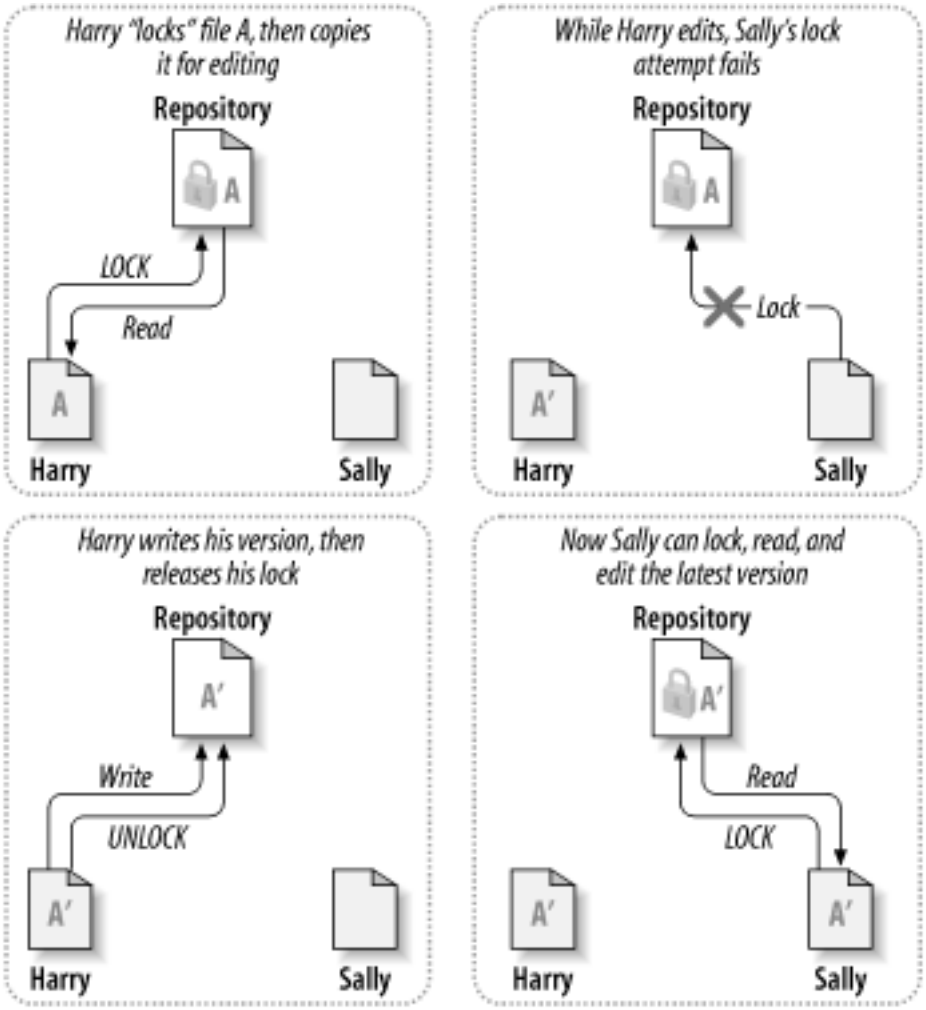
Figura 2.2. El problema a evitar



La solución bloqueo-modificación-desbloqueo

Muchos sistemas de control de versiones utilizan un modelo de *bloqueo-modificación-desbloqueo* para atacar este problema. En un sistema como éste, el repositorio sólo permite a una persona modificar un archivo al mismo tiempo. Juan debe “bloquear” primero el archivo para luego empezar a hacerle cambios. Bloquear un archivo se parece mucho a pedir prestado un libro de la biblioteca; si Juan ha bloqueado el archivo, entonces Carmen no puede hacerle cambios. Por consiguiente, si ella intenta bloquear el archivo, el repositorio rechazará la petición. Todo lo que puede hacer es leer el archivo y esperar a que Juan termine sus cambios y deshaga el bloqueo. Tras desbloquear Juan el archivo, Carmen puede aprovechar su turno bloqueando y editando el archivo. La figura Figura 2.3, “La solución bloqueo-modificación-desbloqueo” demuestra esta sencilla solución.

Figura 2.3. La solución bloqueo-modificación-desbloqueo



El problema con el modelo bloqueo-modificación-desbloqueo es que es un tanto restrictivo y a menudo se convierte en un obstáculo para los usuarios:

- *Bloquear puede causar problemas administrativos.* En ocasiones Juan bloqueará un archivo y se olvidará de él. Mientras tanto, como Carmen está aún esperando para editar el archivo, sus manos están atadas. Y luego Juan se va de vacaciones. Ahora Carmen debe conseguir que un administrador deshaga el bloqueo de Juan. La situación termina causando muchas demoras innecesarias y pérdida de tiempo.

- *Bloquear puede causar una serialización innecesaria.* ¿Qué sucede si Juan está editando el inicio de un archivo de texto y Carmen simplemente quiere editar el final del mismo archivo? Estos cambios no se solapan en absoluto. Ambos podrían editar el archivo simultáneamente sin grandes perjuicios, suponiendo que los cambios se combinaran correctamente. No hay necesidad de turnarse en esta situación.

- *Bloquear puede causar una falsa sensación de seguridad.* Imaginemos que Juan bloquea y edita el archivo A, mientras que Carmen bloquea y edita el archivo B al mismo tiempo. Pero suponga que A y B dependen uno del otro y que los cambios hechos a

cada uno de ellos son semánticamente incompatibles. Súbitamente A y B ya no funcionan juntos. El sistema de bloqueo se mostró ineficaz a la hora de evitar el problema—sin embargo, y de algún modo, ofreció una falsa sensación de seguridad. Es fácil para Juan y Carmen imaginar que al bloquear archivos, cada uno está empezando una tarea segura y aislada, lo cual les inhibe de discutir sus cambios incompatibles desde un principio.

La solución copiar-modificar-mezclar

Subversion, CVS y otros sistemas de control de versiones utilizan un modelo del tipo *copiar-modificar-mezclar* como alternativa al bloqueo. En este modelo, el cliente de cada usuario se conecta al repositorio del proyecto y crea una *copia de trabajo* personal—una réplica local de los archivos y directorios del repositorio. Los usuarios pueden entonces trabajar en paralelo, modificando sus copias privadas. Finalmente, todas las copias privadas se combinan (o mezclan) en una nueva versión final. El sistema de control de versiones a menudo ayuda con la mezcla, pero en última instancia es un ser humano el responsable de hacer que ésto suceda correctamente.

He aquí un ejemplo. Digamos que Juan y Carmen crean sendas copias de trabajo del mismo proyecto, extraídas del repositorio. Ambos trabajan concurrentemente y hacen cambios a un mismo archivo A dentro de sus copias. Carmen guarda sus cambios en el repositorio primero. Cuando Juan intenta guardar sus cambios más tarde, el repositorio le informa de que su archivo A está *desactualizado*. En otras palabras, que el archivo A en el repositorio ha sufrido algún cambio desde que lo copió por última vez. Por tanto, Juan le pide a su cliente que *mezcle* cualquier cambio nuevo del repositorio con su copia de trabajo del archivo A. Es probable que los cambios de Carmen no se solapen con los suyos; así que una vez que tiene ambos juegos de cambios integrados, Juan guarda su copia de trabajo de nuevo en el repositorio. Las figuras Figura 2.4, “La solución copiar-modificar-mezclar” y Figura 2.5, “La solución copiar-modificar-mezclar (continuación)” muestran este proceso.

Figura 2.4. La solución copiar-modificar-mezclar

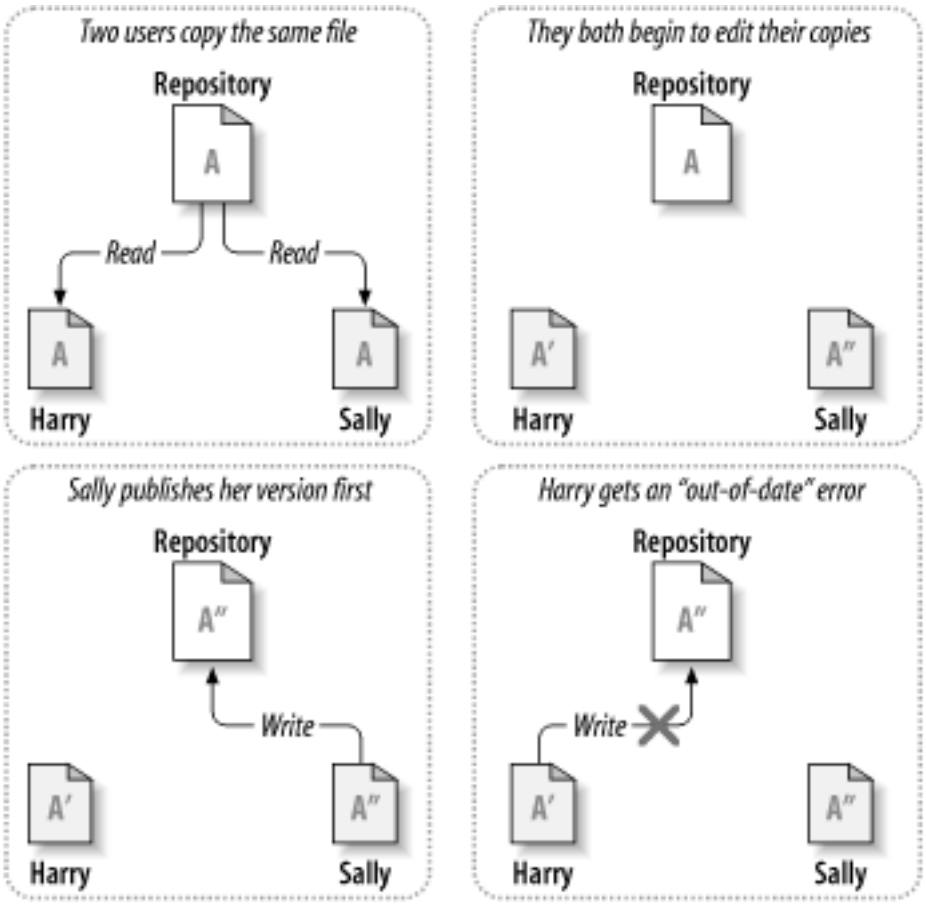
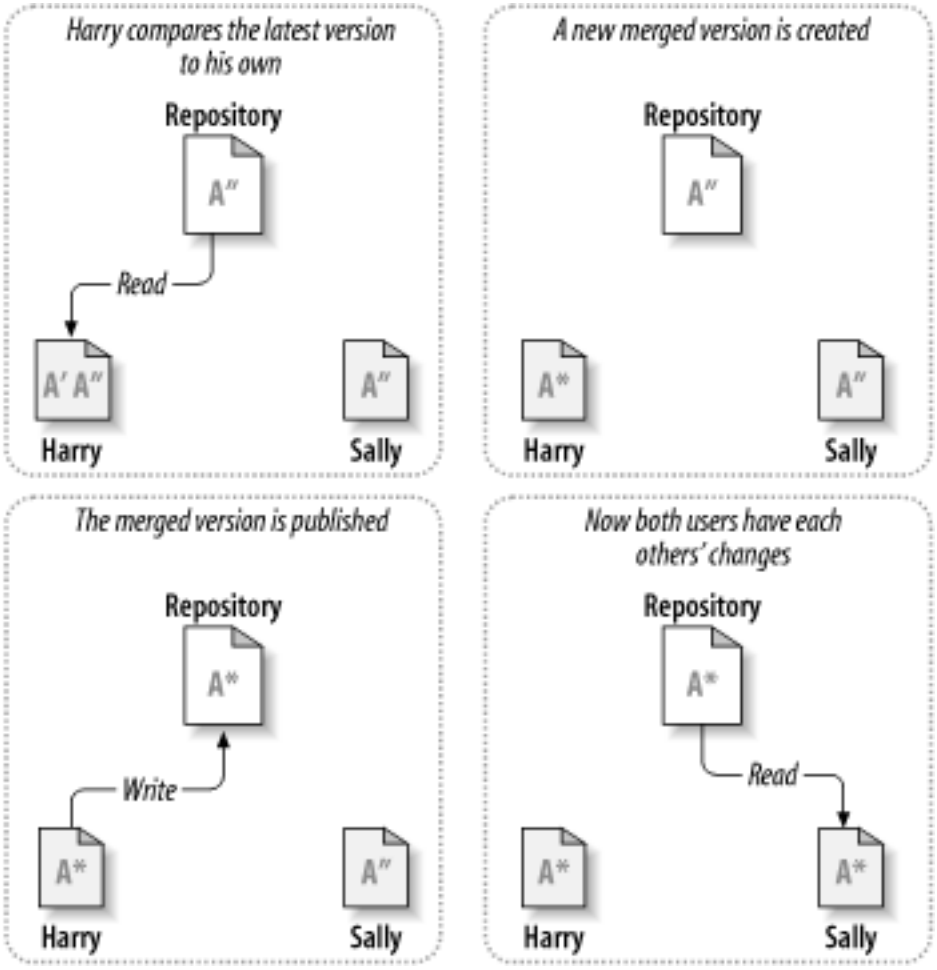


Figura 2.5. La solución copiar-modificar-mezclar (continuación)



¿Pero qué ocurre si los cambios de Carmen *sí* se solapan con los de Juan? ¿Entonces qué? Esta situación se conoce como *conflicto* y no suele suponer un gran problema. Cuando Juan le pide a su cliente que mezcle los últimos cambios del repositorio en su copia de trabajo, su copia del archivo A se marca de algún modo para indicar que está en estado de conflicto: Juan podrá ver ambos conjuntos de cambios conflictivos y escoger manualmente entre ellos. Observe que el programa no puede resolver automáticamente los conflictos; sólo los humanos son capaces de entender y tomar las decisiones inteligentes oportunas. Una vez que Juan ha resuelto manualmente los cambios solapados—posiblemente después de discutirlos con Carmen—ya podrá guardar con seguridad el archivo mezclado en el repositorio.

La solución copiar-modificar-mezclar puede sonar un tanto caótica, pero en la práctica funciona extremadamente bien. Los usuarios pueden trabajar en paralelo, sin tener que esperarse el uno al otro. Cuando trabajan en los mismos archivos, sucede que la mayoría de sus cambios concurrentes no se solapan en absoluto; los conflictos son poco frecuentes. El tiempo que toma resolver los conflictos es mucho menor que el tiempo perdido por un sistema de bloqueos.

Al final, todo desemboca en un factor crítico: la comunicación entre los usuarios. Cuando los usuarios se comunican pobremente, los conflictos tanto sintácticos como semánticos aumentan. Ningún sistema puede forzar a los usuarios a comunicarse perfectamente, y ningún sistema puede detectar conflictos semánticos. Por consiguiente, no tiene sentido dejarse adormecer por la falsa promesa de que un sistema de bloqueos evitará de algún modo los conflictos; en la práctica, el bloqueo parece inhibir la productividad más que otra cosa.

Subversion en acción

Es hora de movernos de lo abstracto a lo concreto. En esta sección mostraremos ejemplos reales de Subversion en la práctica.

Copias de trabajo

Ya ha leído acerca de las copias de trabajo; ahora demostraremos cómo las crea y las usa el cliente de Subversion.

Una copia de trabajo de Subversion es un árbol de directorios corriente de su sistema de archivos local, conteniendo una colección de archivos. Usted puede editar estos archivos del modo que prefiera y si se trata de archivos de código fuente, podrá compilar su programa a partir de ellos de la manera habitual. Su copia de trabajo es su área de trabajo privada: Subversion nunca incorporará los cambios de otra gente o pondrá a disposición de otros sus cambios hasta que usted le indique explícitamente que lo haga.

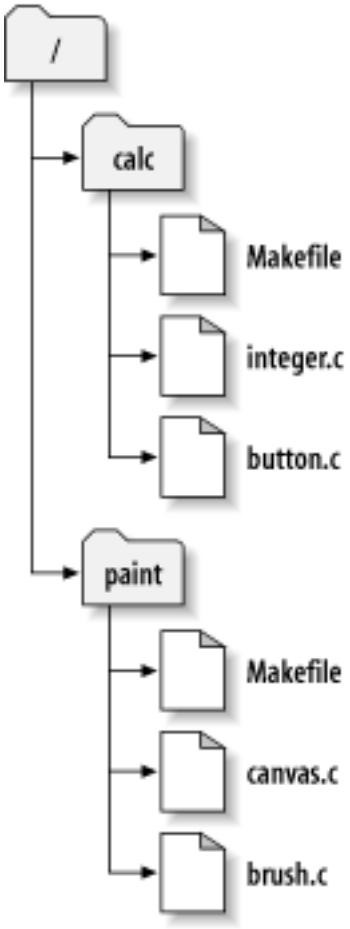
Tras hacer algunos cambios a los archivos en su copia de trabajo y verificar que funcionan correctamente, Subversion le proporciona comandos para "publicar" sus cambios al resto de personas que trabajan con usted en su proyecto (escribiendo en el repositorio). Si las demás personas publican sus propios cambios, Subversion le proporciona comandos para mezclar estos cambios en su directorio de trabajo (leyendo del repositorio).

Una copia de trabajo también contiene algunos archivos extra, creados y mantenidos por Subversion para ayudarle a ejecutar estos comandos. En particular, cada directorio de su copia de trabajo contiene un subdirectorio llamado `.svn`, también conocido como el *directorio administrativo* de la copia de trabajo. Los archivos en cada directorio administrativo ayudan a Subversion a reconocer qué archivos contienen cambios no publicados y qué archivos están desactualizados con respecto al trabajo hecho por los demás.

Un repositorio típico de Subversion contiene a menudo los archivos (o el código fuente) de varios proyectos; normalmente, cada proyecto es un subdirectorio en el árbol del sistema de archivos del repositorio. En esta disposición, la copia de trabajo de un usuario se corresponde habitualmente con un subárbol particular del repositorio.

Por ejemplo, suponga que usted tiene un repositorio que contiene dos proyectos de software, `paint` y `calc`. Cada proyecto reside en su propio subdirectorio dentro del directorio raíz, tal como se muestra en Figura 2.6, "El sistema de archivos del repositorio".

Figura 2.6. El sistema de archivos del repositorio



Para conseguir una copia de trabajo, debe ejecutar primero un *check out* de algún subárbol del repositorio. (El término inglés "check out" puede sonar como si tuviera algo que ver con bloquear o reservar recursos, pero no es así; tan sólo crea una copia privada del proyecto para usted). Por ejemplo, si usted hace un check out de `/calc`, obtendrá una copia de trabajo como ésta:

```
$ svn checkout http://svn.example.com/repos/calc
A  calc
A  calc/Makefile
A  calc/integer.c
A  calc/button.c

$ ls -A calc
Makefile  integer.c  button.c  .svn/
```

La lista de letras A indica que Subversion está añadiendo una serie de elementos a su copia de trabajo. Usted ahora tiene una copia personal del directorio `/calc` del repositorio, con una entrada adicional—`.svn`—que contiene la información extra que Subversion necesita, tal y como se mencionó anteriormente.

URLs del repositorio

A los repositorios de Subversion se puede acceder a través de diferentes métodos—en el disco local, o a través de varios protocolos de red. Sin embargo, la ubicación de un repositorio es siempre un URL. La tabla 2-1 describe la correspondencia entre los diferentes esquemas de URL y los métodos de acceso disponibles.

Tabla 2.1. URLs de Acceso al Repositorio

Esquema	Método de acceso
`file:///`	acceso directo al repositorio (en disco local)
`http://`	acceso vía protocolo WebDAV a un servidor Apache que entiende de Subversion
`https://`	igual que `http://`, pero con cifrado SSL.
`svn://`	acceso vía un protocolo personalizado a un servidor `svnserve`.
`svn+ssh://`	igual que `svn://`, pero a través de un túnel SSH.

En general, los URLs de Subversion utilizan la sintaxis estándar, permitiendo la especificación de nombres de servidores y números de puertos como parte del URL. Recuerde que el método de acceso `file:` es válido sólo para ubicaciones en el mismo servidor donde se ejecuta el cliente—de hecho, se requiere por convención que la parte del URL con el nombre del servidor esté ausente o sea `localhost`:

```
$ svn checkout file:///ruta/a/repositorio
…
$ svn checkout file://localhost/ruta/a/repositorio
…
```

Además, los usuarios del esquema `file:` en plataformas Windows necesitarán usar una sintaxis "estándar" extraoficial para acceder a repositorios que están en la misma máquina, pero en una unidad de disco distinta de la que el cliente esté utilizando en el momento. Cualquiera de las dos siguientes sintaxis para rutas de URL funcionarán siendo `X` la unidad donde reside el repositorio:

```
C:\> svn checkout file:///X:/ruta/a/repositorio
…
C:\> svn checkout "file:///X|/ruta/a/repositorio"
…
```

En la segunda sintaxis, es necesario encerrar el URL entre comillas para que la barra vertical no sea interpretada como una tubería.

Nótese que un URL usa barras de separación ordinarias aún cuando la forma de ruta nativa (no para URLs) en Windows utiliza barras invertidas.

Suponga que hace cambios a `button.c`. Puesto que el directorio `.svn` recuerda la fecha de modificación del archivo y su contenido original, Subversion es capaz de darse cuenta de que el archivo ha cambiado. Sin embargo, Subversion no hará públicos sus cambios hasta que usted no le diga explícitamente que lo haga. El acto de publicar sus cambios es conocido comúnmente como *consignar* (o *registrar*) los cambios al repositorio.

Para publicar sus cambios a otros, usted puede utilizar el comando **commit** de Subversion:

```
$ svn commit button.c
Sending        button.c
Transmitting file data .
Committed revision 57.
```

Ahora sus cambios a `button.c` han sido consignados al repositorio; si otro usuario obtiene una copia de trabajo de `/calc`, podrá ver sus cambios en la última versión del archivo.

Suponga que tiene un colaborador, Carmen, quien obtuvo una copia de trabajo de `/calc` al mismo tiempo que usted. Cuando usted envía sus cambios sobre `button.c`, la copia de trabajo de Carmen se deja sin cambios; Subversion solo modifica las copias de trabajo a petición del usuario.

Para tener su proyecto actualizado, Carmen puede pedir a Subversion que proceda a *actualizar* su copia de trabajo, usando para ello el comando **update** de Subversion. Ésto incorporará los cambios hechos por usted en la copia de trabajo de Carmen, así como otros cambios consignados desde que ella hizo el check out.

```
$ pwd
/home/sally/calc

$ ls -A
.svn/ Makefile integer.c button.c

$ svn update
U button.c
```

La salida del comando **svn update** indica que Subversion actualizó el contenido de `button.c`. Observe que Carmen no necesitó especificar qué archivos actualizar; Subversion usa la información del directorio `.svn`, junto con información adicional del repositorio, para decidir qué archivos necesitan una actualización.

Revisiones

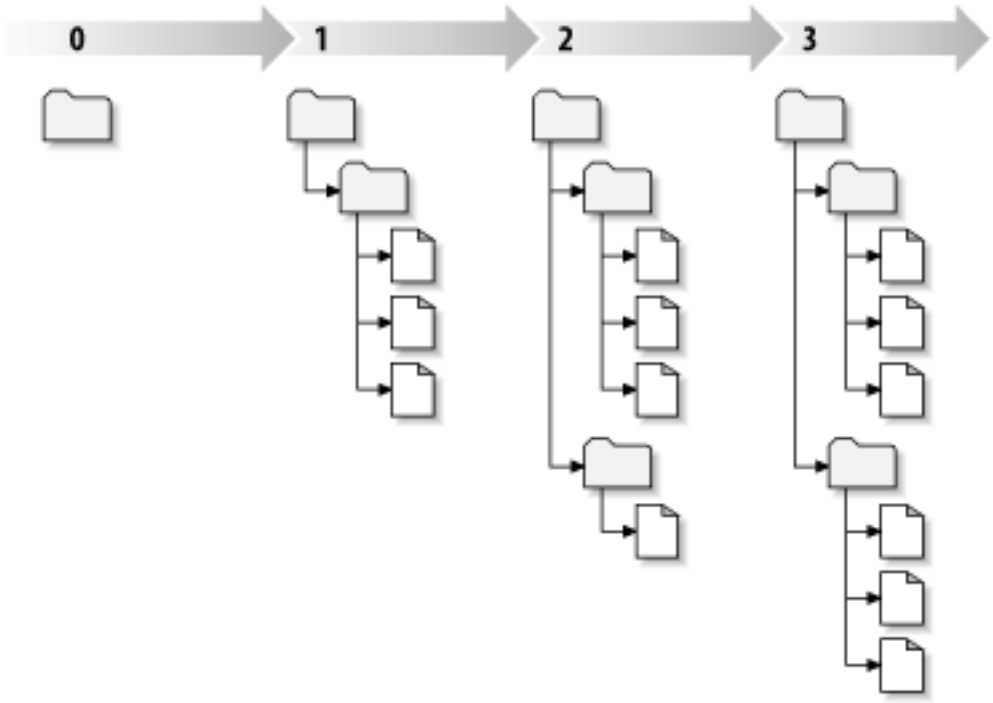
Una operación **svn commit** puede publicar los cambios sobre cualquier número de ficheros y directorios como una única transacción atómica. En su copia privada, usted puede cambiar el contenido de los ficheros, crear, borrar, renombrar y copiar ficheros y directorios, y luego enviar el conjunto entero de cambios como si se tratara de una unidad.

En el repositorio, cada cambio es tratado como una transacción atómica: o bien se realizan todos los cambios, o no se realiza ninguno. Subversion trata de conservar esta atomicidad para hacer frente a posibles fallos del programa, fallos del sistema, problemas con la red, y otras acciones del usuario.

Cada vez que el repositorio acepta un envío, éste da lugar a un nuevo estado del árbol de ficheros llamado *revisión*. A cada revisión se le asigna un número natural único, una unidad mayor que el número de la revisión anterior. La revisión inicial de un repositorio recién creado se numera con el cero, y consiste únicamente en un directorio raíz vacío.

La Figura 2.7, “El repositorio” ilustra una manera interesante de ver el repositorio. Imagine un array de números de revisión, comenzando por el 0, que se extiende de izquierda a derecha. Cada número de revisión tiene un árbol de ficheros colgando debajo de él, y cada árbol es una “instantánea” del aspecto del repositorio tras cada envío.

Figura 2.7. El repositorio



Números de revisión global

A diferencia de muchos otros sistemas de control de versiones, los números de revisión de Subversion se aplican a *árboles enteros*, no a ficheros individuales. Cada número de revisión selecciona un árbol completo, un estado particular del repositorio tras algún cambio publicado. Otra manera de ver ésto es que la revisión N representa el estado del sistema de ficheros del repositorio tras el envío de cambios N-ésimo. Cuando un usuario de Subversion habla de la "revisión 5 de `foo.c`", lo que realmente quiere decir es "`foo.c` tal como aparece en la revisión 5." Observe que en general, ¡las revisiones N y M de un fichero *no* tienen por qué ser diferentes necesariamente! Dado que CVS utiliza números de revisión para cada fichero, los usuarios de CVS pueden querer echar un vistazo a Apéndice A, *Subversion para usuarios de CVS* para más detalles.

Es importante observar que las copias de trabajo no siempre se corresponden con una revisión en particular del repositorio; pueden contener ficheros de varias revisiones diferentes. Por ejemplo, suponga que obtiene una copia de trabajo de un repositorio cuya revisión más reciente es la 4:

```
calc/Makefile:4
     integer.c:4
     button.c:4
```

Por el momento, esta copia de trabajo se corresponde exactamente con la revisión 4 del repositorio. Sin embargo, suponga que realiza un cambio a `button.c` y lo publica. Suponiendo que no se han realizado otros envíos, el suyo creará la revisión 5 del repositorio, y su copia de trabajo aparecerá ahora así:

```
calc/Makefile:4
     integer.c:4
     button.c:5
```

Suponga que, en este punto, Carmen envía un cambio a `integer.c`, creando la revisión 6. Si usted usa **svn update** para actualizar su copia de trabajo, ésta se verá ahora como:

```
calc/Makefile:6
     integer.c:6
     button.c:6
```

Los cambios de Carmen sobre `integer.c` aparecerán en su copia de trabajo y las modificaciones hechas por usted seguirán presentes en `button.c`. En este ejemplo, el texto de `Makefile` es idéntico en las revisiones 4, 5 y 6, aunque Subversion marcará su copia de trabajo de Makefile con la revisión 6 para indicar que ya está actualizada. Por lo tanto, después de hacer una actualización limpia en el directorio raíz de su copia de trabajo, ésta se corresponderá generalmente con una revisión del repositorio exactamente.

Cómo las copias de trabajo siguen la pista al repositorio

Para cada fichero de una copia de trabajo, Subversion registra dos datos esenciales en el área administrativa `.svn/`:

- revisión en la que está basado el fichero de la copia de trabajo (ésto se llama la *revisión de trabajo* del fichero), y
- una marca de tiempo con la fecha de la última actualización del fichero desde el repositorio.

Con esta información, y comunicándose con el repositorio, Subversion puede conocer en cuál de los cuatro estados siguientes se encuentra el fichero de la copia de trabajo:

Sin cambios y actualizado

> El fichero no ha sido modificado en la copia de trabajo ni se ha enviado ningún cambio sobre ese fichero al repositorio desde su revisión de trabajo. Un **svn commit** de ese fichero no hará nada, y un **svn update** del fichero tampoco hará nada.

Modificado localmente y actualizado

> El fichero ha sido modificado en la copia de trabajo pero no se ha enviado ningún cambio sobre ese fichero al repositorio desde su revisión base. Hay cambios locales que no han sido enviados al repositorio, por lo que un **svn commit** del fichero publicará con éxito sus cambios, y un **svn update** del fichero no hará nada.

Sin cambios y desactualizado

> El fichero no ha sido modificado en la copia de trabajo, pero sí en el repositorio. El fichero debería ser actualizado para sincronizarlo con la revisión pública. Un **svn commit** del fichero no hará nada, y un **svn update** del fichero introducirá los últimos cambios en su copia de trabajo.

Modificado localmente y desactualizado

> El fichero ha sido modificado tanto en la copia de trabajo como en el repositorio. Un **svn commit** del fichero fallará dando un error de “desactualizado”. El fichero debe ser actualizado primero; un **svn update** intentará mezclar los cambios públicos con los cambios locales. Si Subversion no puede combinar los cambios de manera convincente automáticamente, dejará que sea el usuario el que resuelva el conflicto.

Todo ésto puede parecer un montón de cosas a tener en cuenta, pero el comando **svn status** le mostrará el estado de cualquier elemento de su copia de trabajo. Para obtener más información acerca de ese comando, vea **“svn status”**.

Las limitaciones de las revisiones mixtas

Por norma general, Subversion trata de ser tan flexible como sea posible. Un tipo especial de flexibilidad es la habilidad para tener dentro de una copia de trabajo números de revisión mixtos.

Para comenzar, puede que no esté completamente claro el por qué este tipo de flexibilidad se considera una característica y no un problema. Después de completar un envío al repositorio, los ficheros y directorios recién enviados se encuentran en una revisión de trabajo más reciente que el resto de la copia de trabajo. Parece un poco lioso. Tal como se mostró anteriormente, siempre se puede dejar una copia de trabajo en una única revisión de trabajo ejecutando **svn update**. ¿Por qué querría alguien *deliberadamente* tener una mezcla de revisiones de trabajo?

Suponiendo que su proyecto es lo suficientemente complejo, descubrirá que a veces es conveniente forzar la “desactualización” de ciertas partes de su copia de trabajo a una versión anterior; aprenderá cómo hacer ésto en el capítulo 3. Quizás quiera probar una versión anterior de un submódulo contenido en un subdirectorio, o tal vez quiera examinar una serie de versiones previas de un fichero en el contexto del último árbol.

Por mucho que usted haga uso de revisiones mixtas en su copia de trabajo, hay ciertas limitaciones asociadas a esta flexibilidad.

Primero, usted no puede publicar la eliminación de un fichero o directorio que no esté completamente actualizado. Si existe una versión más reciente en el repositorio, su intento de eliminación será rechazado para impedir que destruya accidentalmente cambios que aún no ha visto.

Segundo, usted no puede publicar los cambios en los metadatos de un directorio a menos que esté completamente actualizado. Aprenderá cómo adjuntar “propiedades” a elementos en el capítulo 6. Una revisión de trabajo de un directorio define un conjunto específico de entradas y propiedades, y por tanto enviar un cambio a una propiedad de un directorio desactualizado puede destruir las propiedades que no haya visto todavía.

Resumen

A lo largo de este capítulo hemos tratado una serie de conceptos fundamentales acerca de Subversion:

- Hemos introducido las nociones de repositorio central, la copia de trabajo del cliente, y el array de árboles de revisiones del repositorio.
- Hemos visto algunos ejemplos sencillos de cómo dos colaboradores pueden usar Subversion para publicar y recibir cambios uno del otro usando el modelo 'copiar-modificar-mezclar'.
- Hemos hablado un poco sobre la manera en que Subversion sigue y maneja la información de una copia de trabajo.

A estas alturas usted ya debería tener una idea más o menos clara de cómo funciona Subversion a nivel general. Armado con este conocimiento, debería estar listo para pasar al siguiente capítulo, el cual es un recorrido detallado por los comandos y características de Subversion.

Capítulo 3. Recorrido guiado

Ahora entraremos en los detalles de usar Subversion. Para el momento en que usted alcance el final de este capítulo, podrá realizar casi todas las tareas necesarias para usar Subversion de manera cotidiana. Comenzará con una descarga inicial de su código, e irá haciendo cambios y examinando esos cambios. Usted también verá como introducir cambios hechos por otros en su copia de trabajo, examinarlos, y resolver cualquier conflicto que pudiera surgir.

Observe que este capítulo no pretende ser una lista exhaustiva de los comandos de Subversion —más bien es una introducción informal a las tareas más comunes de Subversion que se encontrará. Este capítulo asume que usted ha leído y entendido Capítulo 2, *Conceptos básicos* y está familiarizado con el modelo general de Subversion. Para una referencia completa de todos los comandos, vea Capítulo 9, *Referencia completa de Subversion*.

¡Ayuda!

Antes de seguir leyendo, aquí está el comando más importante que usted necesitará cuando esté usando Subversion: **svn help**. El cliente de línea de comandos Subversion está auto-documentado— en cualquier momento, un **svn help <subcomando>** rápido describirá la sintaxis, las opciones y el comportamiento del **subcomando**.

Import

Use **svn import** para importar un nuevo proyecto dentro de un repositorio Subversion. Mientras que esto es muy probablemente lo primero que usted hará cuando instale su servidor Subversion, no es algo que pase muy a menudo. Para una descripción detallada de import, vea "**svn import**" más adelante en este capítulo.

Revisiones: Números, Palabras Clave, y Fechas, ¡Dios Mío!

Antes de continuar, usted debería saber un poco acerca de como identificar una revisión particular en su repositorio. Como aprendió en "Revisiones", una revisión es una "instantánea" de su repositorio en un momento particular en el tiempo. A medida que continúe depositando y haciendo crecer su repositorio, usted necesita un mecanismo para identificar estas instantáneas.

Usted puede especificar estas revisiones usando `--revision` (`-r`) junto con la revisión que desee (**svn --revision REV**) o puede especificar un rango separando dos revisiones con dos puntos (**svn --revision REV1:REV2**). Y Subversion le permite referirse a estas revisiones por número, palabra clave, o fecha.

Números de revisión

Cuando usted crea un nuevo repositorio de Subversion, este comienza su vida en la revisión cero y cada envío sucesivo incrementa el número de revisión en uno. Después de completar su envío, el cliente de Subversion le informa del nuevo número de revisión:

```
$ svn commit --message "Corrected number of cheese slices."
Sending        sandwich.txt
Transmitting file data .
Committed revision 3.
```

Si en cualquier punto del futuro usted quiere referirse a esa revisión (veremos cómo y por qué podríamos querer hacer esto más adelante en este capítulo), puede referirse a ella como "3".

Palabras clave de la revisión

El cliente de Subversion entiende una cantidad de *palabras clave de la revisión*. Estas palabras clave pueden ser usadas en vez del número entero como argumento a la opción `--revision`, y son resueltos como números específicos de revisión por Subversion:

Cada directorio en su copia de trabajo contiene un subdirectorio administrativo llamado `.svn`. Para todos los ficheros en un directorio, Subversion mantiene una copia de cada fichero en el área administrativa. Esta copia es una copia no modificada (ninguna expansión de palabra clave, ninguna traducción del fin de línea, nada de nada) del fichero tal como existió en la última revisión (llamada la revisión "BASE") que usted actualizó a su copia de trabajo. Nos referimos a este fichero como *copia prístina* o versión *basada en texto* de su fichero, y siempre es una copia byte-a-byte exacta del fichero tal como existe en el repositorio.

HEAD

La última revisión en el repositorio.

BASE

La revisión "prístina" de un elemento en la copia de trabajo.

COMMITTED

La última revisión en la que un elemento cambió antes (o en) BASE.

PREV

La revisión inmediatamente *anterior* a la última revisión en la cual un elemento cambió. (Técnicamente, `COMMITTED - 1`.)

`PREV`, `BASE`, y `COMMITTED` pueden ser usadas para referirse a rutas locales, pero no a URLs.

Aquí hay algunos ejemplos de palabras clave de revisión en acción. No se preocupe si los comandos no tienen sentido todavía; estaremos explicando estos comandos a medida que avancemos por el capítulo:

```
$ svn diff --revision PREV:COMMITTED foo.c
# shows the last change committed to foo.c

$ svn log --revision HEAD
# shows log message for the latest repository commit

$ svn diff --revision HEAD
# compares your working file (with local mods) to the latest version
# in the repository.

$ svn diff --revision BASE:HEAD foo.c
# compares your "pristine" foo.c (no local mods) with the
# latest version in the repository

$ svn log --revision BASE:HEAD
# shows all commit logs since you last updated

$ svn update --revision PREV foo.c
# rewinds the last change on foo.c.
# (foo.c's working revision is decreased.)
```

Estas palabras clave le permiten realizar muchas operaciones comunes (y útiles) sin tener que buscar números específicos de revi-

sión o recordar la revisión exacta de su copia de trabajo.

Fechas de revisión

Dondequiera que usted especifique un número de revisión o palabra clave de revisión, usted también puede especificar una fecha dentro de llaves “{}”. ¡Puede incluso tener acceso a un rango de cambios en el repositorio usando fechas y revisiones juntas!

Aquí hay ejemplos de los formatos de fecha que admite Subversion. Recuerde usar comillas alrededor de cualquier fecha que contenga espacios.

```
$ svn checkout --revision {2002-02-17}
$ svn checkout --revision {15:30}
$ svn checkout --revision {15:30:00.200000}
$ svn checkout --revision {"2002-02-17 15:30"}
$ svn checkout --revision {"2002-02-17 15:30 +0230"}
$ svn checkout --revision {2002-02-17T15:30}
$ svn checkout --revision {2002-02-17T15:30Z}
$ svn checkout --revision {2002-02-17T15:30-04:00}
$ svn checkout --revision {20020217T1530}
$ svn checkout --revision {20020217T1530Z}
$ svn checkout --revision {20020217T1530-0500}
…
```

Cuando usted especifica una fecha como revisión, Subversion encuentra la revisión más reciente en el repositorio a esa fecha:

```
$ svn log --revision {2002-11-28}
------------------------------------------------------------------------
r12 | ira | 2002-11-27 12:31:51 -0600 (Wed, 27 Nov 2002) | 6 lines
…
```

¿Está Subversion adelantado un día?

Si usted especifica una fecha simple como revisión sin especificar una hora del día (por ejemplo `2002-11-27`), usted podría pensar que Subversion debería darle la última revisión que tuvo lugar el 27 de Noviembre. En cambio, usted recibirá una revisión del 26 de Noviembre, o incluso anterior. Recuerde que Subversion encontrará la *revisión más reciente del repositorio* a la fecha que usted da. Si da una fecha sin una marca de tiempo, como `2002-11-27`, Subversion asume la hora como 00:00:00, así que buscar la revisión más reciente no devolverá nada del día 27.

Si quiere incluir el día 27 en su búsqueda, puede especificar el día 27 con la hora (`{"2002-11-27 23:59"}`), o simplemente especificar el día siguiente (`{2002-11-28}`).

También puede especificar un rango de fechas. Subversion encontrará todas las revisiones entre ambas fechas, inclusive:

```
$ svn log --revision {2002-11-20}:{2002-11-29}
…
```

Como señalamos, también puede mezclar fechas y revisiones:

```
$ svn log --revision {2002-11-20}:4040
```

Los usuarios deberían estar enterados de una sutileza que puede convertirse en un obstáculo al tratar con fechas en Subversion. Co-

mo la marca de tiempo de una revisión es guardada como una propiedad de la revisión— una propiedad sin versionar, modificable — las marcas de tiempo de las revisiones pueden cambiarse para representar una falsificación completa de la verdadera cronología, o incluso borrarla totalmente. Esto causará estragos en la conversión interna fecha-a-revisión que realiza Subversion.

Descarga inicial

La mayor parte del tiempo, usted empezará a usar un repositorio de Subversion haciendo un *checkout* de su proyecto . Descargar un repositorio crea una copia de éste en en su máquina local. Esta copia contiene el HEAD (última revisión) del repositorio de Subversion que usted especifica en la línea de comandos:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk
A  trunk/subversion.dsw
A  trunk/svn_check.dsp
A  trunk/COMMITTERS
A  trunk/configure.in
A  trunk/IDEAS
…
Checked out revision 2499.
```

Esquema del repositorio

Si se está preguntando qué es `trunk` en la URL anterior, es parte de la manera que le recomendamos estructurar su repositorio Subversion, acerca del cual hablaremos mucho más en Capítulo 4, *Crear ramas y fusionarlas*.

Aunque el ejemplo de arriba descarga el directorio trunk, usted puede descargar fácilmente cualquier subdirectorio más profundo de un repositorio especificando el subdirectorio en la URL de descarga:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk/doc/book/tools
A  tools/readme-dblite.html
A  tools/fo-stylesheet.xsl
A  tools/svnbook.el
A  tools/dtd
A  tools/dtd/dblite.dtd
…
Checked out revision 2499.
```

Como Subversion usa un modelo “copie-modifique-fusione” en vez de “bloquear-modificar-desbloquear” (vea Capítulo 2, *Conceptos básicos*), usted ya puede empezar a realizar cambios a los ficheros y directorios en su copia de trabajo local. Su copia local es justo como cualquier otra colección de ficheros y directorios en su sistema. Usted puede editarlos y cambiarlos, moverlos, usted puede incluso borrar la copia local entera y olvidarse de ella.

Mientras que su copia de trabajo local es “justo como cualquier otra colección de ficheros y directorios en su sistema”, usted necesita hacer saber a Subversion si va a reacomodar cualquier cosa dentro de su copia local. Si desea copiar o mover un elemento en una copia local, debe usar **svn copy** o **svn move** en vez de los comandos para copiar y mover proporcionados por su sistema operativo. Hablaremos más acerca de ellos al avanzar en este capítulo.

A menos que esté listo para enviar al repositorio un fichero o directorio nuevo, o cambios a unos existentes, no hay necesidad de indicarle al servidor de Subversion que usted haya hecho algo más.

¿Qué pasa con el directorio .svn?

Cada directorio en una copia de trabajo local contiene un área administrativa, un subdirectorio llamado `.svn`. Generalmente, los comandos de listado de directorios no muestran este subdirectorio, pero este es sin embargo un directorio importante. Sea lo que fuere lo que haga ¡no borre o cambie nada en el área administrativa! Subversion depende de ella para administrar su copia de trabajo local.

Mientras que usted puede descargar una copia de trabajo local con la URL del repositorio como único argumento, también puede especificar un directorio después de su URL del repositorio. Esto pone su copia de trabajo local dentro del nuevo directorio que usted nombra. Por ejemplo:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subv
A  subv/subversion.dsw
A  subv/svn_check.dsp
A  subv/COMMITTERS
A  subv/configure.in
A  subv/IDEAS
…
Checked out revision 2499.
```

Esto pondrá su copia de trabajo local en un directorio llamado `subv` en vez de en un directorio llamado `trunk` como hicimos previamente.

Ciclo básico de trabajo

Subversion tiene numerosas características, opciones, campanas y silbidos, pero bajo uso cotidiano lo más probable es que use solamente algunas de ellas. En esta sección veremos las cosas más comunes que usted puede encontrarse haciendo con Subversion en el transcurso de un día de trabajo.

El ciclo de trabajo típico se parece a esto:

- Actualizar su copia de trabajo local
  - **svn update**
- Hacer cambios
  - **svn add**
  - **svn delete**
  - **svn copy**
  - **svn move**
- Examinar sus cambios
  - **svn status**

- **svn diff**
- **svn revert**

- Fusionar los cambios de otros en su copia de trabajo
  - **svn merge**
  - **svn resolved**

- Enviar sus cambios
  - **svn commit**

Actualizar su copia de trabajo local

Cuando se trabaja en un proyecto con un equipo, usted querrá actualizar su copia de trabajo local para recibir cualquier cambio hecho desde su última actualización por otros desarrolladores en el proyecto. Use **svn update** para poner a su copia de trabajo local en sincronía con la última revisión en el repositorio.

```
$ svn update
U  foo.c
U  bar.c
Updated to revision 2.
```

En este caso, alguien envió modificaciones a `foo.c` y `bar.c` desde la última vez que usted actualizó, y Subversion ha actualizado su copia de trabajo local para incluir estos cambios.

Vamos a examinar la salida de **svn update** un poco más. Cuando el servidor envía cambios a su copia de trabajo local, un código de letras es mostrado al lado de cada elemento para hacerle saber qué acciones realizó Subversion para actualizar su copia de trabajo local:

`U foo`

El fichero `foo` fue actualizado[1] (recibidos cambios del servidor).

`A foo`

El fichero o directorio `foo` fue Añadido a su copia de trabajo local.

`D foo`

El fichero o directorio `foo` fue borrado [2] de su copia de trabajo local.

`R foo`

[1] N. del T.: La inicial usada representa la palabra Updated (actualizado) en el idioma inglés
[2] N. del T.: La inicial usada representa la palabra Deleted (borrado) en el idioma inglés

El fichero o directorio `foo` fue Reemplazado en su copia de trabajo local; esto es, `foo` fue borrado, y un nuevo objeto con el mismo nombre fue añadido. Aunque pueden tener el mismo nombre, el repositorio los considera objetos distintos con historiales distintos.

`G foo`

El fichero `foo` recibió nuevos cambios del repositorio, pero su copia local del fichero tenía las modificaciones que ud. ya había hecho. O los cambios no se intersectaron , o los cambios eran exactamente iguales que sus modificaciones locales, así que Subversion ha fusionado [3] satisfactoriamente los cambios del repositorio en el fichero sin ningún problema.

`C foo`

El fichero `foo` recibió cambios Conflicting del servidor. Los cambios del servidor directamente se superpusieron sobre sus propios cambios en el fichero. Aunque no hay necesidad de aterrarse. Esta superposición necesita ser resuelta por un humano (usted); tratamos esta situación más tarde en este capítulo.

Hacer cambios en su copia de trabajo local

Ahora puede conseguir trabajar y hacer cambios en su copia de trabajo local. Generalmente es más conveniente decidir un cambio particular (o un conjunto de cambios) a hacer, por ejemplo escribir una nueva característica, corregir un fallo, etc. Los comandos de Subversion que puede usar aquí son **svn add**, **svn delete**, **svn copy**, y **svn move**. Sin embargo, si usted está simplemente corrigiendo los archivos que están ya en Subversion, puede no necesitar utilizar ninguno de estos comandos hasta que envíe los cambios. Cambios que puede hacer a su copia de trabajo local:

Cambios en los ficheros

Esta es la clase más simple de cambio. No necesita decirle a Subversion que se propone a cambiar un fichero; simplemente haga los cambios. Subversion será capaz de detectar automáticamente qué archivos han sido cambiados.

Cambios en el árbol

Puede preguntar a Subversion que “marque” ficheros y directorios para el borrado planificado, la adición, la copia, o moverlos. Mientras estos cambios pueden ocurrir inmediatamente en su copia de trabajo local, ninguna adición o borrado sucederá en el repositorio hasta que envíe los cambios.

Para hacer cambios en los ficheros, use su editor de textos, procesador de textos, programa de gráficos , o cualquier herramienta que usaría normalmente. Subversion maneja ficheros binarios tan fácilmente como maneja archivos de texto—y tan eficientemente también.

Aquí hay una descripción de los cuatro subcomandos de Subversion que usted usará más a menudo para hacer cambios del árbol (cubriremos **svn import** y **svn mkdir** después.

svn add foo

Programa añadir `foo` al repositorio. Cuando haga su próximo envío, `foo` se convertirá en hijo de su directorio padre. Fíjese que si `foo` es un directorio, todo por debajo de `foo` será programado para la adición . Si solo quiere añadir el propio `foo`, pase la opción `--non-recursive` (`-N`) .

svn delete foo

Programa borrar `foo` del repositorio. Si `foo` es un fichero, se borrará inmediatamente de su copia de trabajo local. Si `foo` es un directorio, este no es borrado, pero Subversion lo programa para borrarlo. Cuando envíe sus cambios, `foo` será borrado de su copia de trabajo y del repositorio.[4]

[3] N. del T.: merGed, fusionado, en inglés

svn copy foo bar

Crea un nuevo objeto `bar` como duplicado de `foo`. `bar` es automáticamente programado para la adición. Cuando `bar` es añadido al repositorio en el siguiente envío de cambios, su historia de copia es registrada (como que originalmente viene de `foo`).**svn copy** no crea directorios intermedios.

svn move foo bar

Este comando funciona exactamente igual que **svn copy foo bar; svn delete foo**. Esto es, se programa `bar` para la adición como una copia de `foo`, y se programa `foo` para la eliminación. **svn move** no crea directorios intermedios.

Cambiando el repositorio sin una copia de trabajo

Anteriormente en este capítulo, dijimos que tiene que enviar cualquier cambio que usted haga en orden para que el repositorio refleje estos cambios. Esto no es enteramente cierto —*hay* algunos usos-casos que inmediatamente envían los cambios del árbol al repositorio. Esto sucede solamente cuando un subcomando está operando directamente sobre una URL, al contrario que sobre una ruta de la copia-local . En particular, usos específicos de **svn mkdir**, **svn copy**, **svn move**, y **svn delete** pueden trabajar con URLs.

Las operaciones de URL se comportan de esta manera porque los comandos que manipulan en una copia de trabajo pueden usar la copia de trabajo como una clase de “área de estancamiento”para preparar sus cambios antes de enviarlos al repositorio. Los comandos que operan sobre URLs no tienen este lujo, así cuando usted opera directamente sobre una URL, cualquiera de las acciones anteriores representa un envío inmediato.

Examine sus cambios

Una vez que haya terminado de hacer cambios, necesita enviarlos al repositorio, pero antes de hacerlo, generalmente es buena idea echar un vistazo a lo que ha cambiado exactamente. Examinando sus cambios antes de que los envíe, puede hacer un mensaje de informe de cambios más exacto. También puede descubrir que ha cambiado inadvertidamente un fichero, y esto le da la posibilidad de invertir esos cambios antes de enviarlos. Además, esta es una buena oportunidad para revisar y escudriñar cambios antes de publicarlos. Usted puede ver exactamente qué cambios ha hecho usando **svn status**, **svn diff**, y **svn revert**. Generalmente usará los primeros dos comandos para descubrir qué ficheros han cambiado en su copia de trabajo local, y después quizá el tercero para invertir algunos (o todos) de esos cambios.

Subversion ha sido optimizado para ayudarle con esta tarea, y es capaz de hacer muchas cosas sin comunicarse con el repositorio. En particular, su copia de trabajo local contiene una copia “prístina” secreta almacenada de cada fichero de versión controlado dentro del área `.svn`. Debido a esto, Subversion puede rápidamente mostrarle cómo sus ficheros de trabajo han cambiado, o incluso permitirle deshacer sus cambios sin contactar con el repositorio.

svn status

Probablemente usará el comando **svn status** más que cualquier otro comando de Subversion.

Usuarios de CVS: ¡Lleve a cabo esa actualización!

Probablemente esté acostumbrado a usar **cvs update** para ver qué cambios le ha hecho a su copia de trabajo local. **svn status** le dará toda la información que necesita sobre qué ha cambiado en su copia de trabajo local—sin acceder al repositorio o incorporando nuevos cambios potenciales publicados por otros usuarios.

En Subversion, **update** hace justo eso—actualiza su copia de trabajo local con cualquier cambio enviado al repositorio desde la última vez que usted ha actualizado su copia de trabajo. Tendrá que romper el hábito de usar el comando **update** para ver

[4] Por supuesto, nada es borrado totalmente del repositorio—solo del`HEAD` del repositorio. Puede conseguir cualquier cosa que borró descargando (o actualizando su copia de trabajo) a una revisión anterior a la que lo borró.

qué modificaciones locales ha hecho.

Si ejecuta **svn status** en lo alto de su copia de trabajo sin argumentos, detectará todos los cambios de fichero y árbol que usted ha hecho. Este ejemplo está diseñado para mostrar todos los códigos de estado diferentes que **svn status** puede devolver. (Observe que el texto que sigue a # en el siguiente ejemplo no es impreso realmente por **svn status**.)

```
$ svn status
  L     abc.c                 # svn has a lock in its .svn directory for abc.c
M       bar.c                 # the content in bar.c has local modifications
 M      baz.c                 # baz.c has property but no content modifications
X       3rd_party             # this dir is part of an externals definition
?       foo.o                 # svn doesn't manage foo.o
!       some_dir              # svn manages this, but it's either missing or incomplete
~       qux                   # versioned as dir, but is file, or vice versa
I       .screenrc             # this file is ignored
A  +    moved_dir             # added with history of where it came from
M  +    moved_dir/README      # added with history and has local modifications
D       stuff/fish.c          # this file is scheduled for deletion
A       stuff/loot/bloo.h     # this file is scheduled for addition
C       stuff/loot/lump.c     # this file has conflicts from an update
    S   stuff/squawk          # this file or dir has been switched to a branch
…
```

En este formato de salida **svn status** impresa cinco columnas de caracteres, seguidos por varios caracteres de espacio en blanco, seguido por un nombre de fichero o directorio. La primera columna dice el estado de un fichero o directorio y/o su contenido. Los códigos impresos aquí son:

`A file_or_dir`

El fichero o directorio `file_or_dir` ha sido programado para la adición en el repositorio.

`C file`

El fichero `file` está en un estado de conflicto. Esto es, los cambios recibidos del servidor durante una actualización se solapan con cambios locales que usted tiene en su copia de trabajo. Debe resolver este conflicto antes de enviar sus cambios al repositorio.

`D file_or_dir`

El fichero o directorio `file_or_dir` ha sido programado para la supresión del repositorio.

`M file`

El contenido del fichero `file` ha sido modificado.

`X dir`

El directorio `dir` está sin versionar, pero está relacionado con una definición externa de Subversion. Para descubrir más acerca de definiciones externas, vea “Repositorios externos”.

`? file_or_dir`

El fichero o directorio `file_or_dir` no está bajo control de versiones. Puede silenciar la marca de pregunta pasando la opción `--quiet` (`-q`) a **svn status**, o o poniendo la característica `svn:ignore` en el directorio padre. Para más información sobre ficheros ignorados, vea “`svn:ignore`”.

`! file_or_dir`

El fichero o directorio `file_or_dir` está bajo el control de versiones pero falta o está de alguna manera incompleto. El objeto puede faltar si se ha borrado usando un comando ajeno a Subversion. En el caso de un directorio, puede estar incompleto si ha interrumpido una descarga o una actualización. Un rápido **svn update** repondrá el fichero o el directorio desde el repositorio, o **svn revert file** restaurará un archivo que falta.

`~ file_or_dir`

El fichero o directorio `file_or_dir` está en el repositorio como un tipo de objeto, pero actualmente está en su copia de trabajo como otro tipo. Por ejemplo, Subversion pudo tener un fichero en el repositorio, pero usted borró el fichero y creó un directorio en su lugar, sin usar los comandos **svn delete** o **svn add**.

`I file_or_dir`

Subversion está "ignorando" el fichero o directorio `file_or_dir`, probablemente porque usted se lo dijo. Para más información sobre ficheros ignorados, vea "`svn:ignore`". Observe que este símbolo solo aparece si le pasa la opción `--no-ignore` a **svn status**.

La segunda columna dice el estado de las propiedades de un fichero o un directorio (vea "Propiedades" para más información sobre propiedades). Si aparece una `M` en la segunda columna, entonces las propiedades han sido modificadas, si no un espacio en blanco será impreso.

La tercera columna solo mostrará un espacio en blanco o una `L` la cual significa que Subversion ha bloqueado el objeto en el área de trabajo `.svn`. Usted 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. Si Subversion no se está ejecutando, entonces probablemente Subversion fue interrumpido y el bloqueo necesita ser eliminado ejecutando **svn cleanup** (más sobre eso más adelante en este capítulo).

La cuarta columna solo mostrará un espacio blanco o un `+` el cual significa que el fichero o directorio está programado para ser añadido o modificado con historial adicional adjunto. Esto ocurre típicamente cuando usted **svn move** o **svn copy** un fichero o directorio. Si usted ve `A  +`, esto significa que el objeto está programado para la adición-con-historial. Este puede ser un fichero, o la raíz de un directorio copiado. `+` significa que el objeto es parte de un subárbol programado para la adición-con-historial, p.e.algún padre fue copiado,and it's just coming along for the ride. `M  +` significa que el objeto es parte de un subárbol programado para la adición-con-historial, *y* este tiene modificaciones locales. Cuando envíe los cambios, primero el padre será añadido-con-historial (copiado), lo que significa que este fichero existirá automáticamente en la copia. Entonces las modificaciones locales serán enviadas al repositorio.

La quinta columna solo mostrará un espacio en blanco o una `S`. Esto significa que el fichero o directorio ha sido movido de la ruta del resto de la copia de trabajo (usando **svn switch**) a una rama.

Si usted pasa una ruta específica a **svn status**, este le dará información acerca de ese objeto solamente:

```
$ svn status stuff/fish.c
D      stuff/fish.c
```

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:

```
$ svn status --verbose
M               44        23    sally     README
                44        30    sally     INSTALL
M               44        20    harry     bar.c
                44        18    ira       stuff
                44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
                44        21    sally     stuff/things
A                0         ?     ?        stuff/things/bloo.h
                44        36    harry     stuff/things/gloo.c
```

Esta es la “forma larga” de salida de **svn status**. La primera columna permanece igual, pero la segunda columna muestra la revisión de trabajo del fichero. La tercera y cuarta columna muestra la revisión en la cuál el objeto cambió por última vez, y quién lo cambió.

Ninguna de las invocaciones anteriores a **svn status** contactaban con el repositorio, trabajan solo localmente comparando los metadatos en el directorio `.svn` con la copia de trabajo local. 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:

```
$ svn status --show-updates --verbose
M      *        44        23    sally     README
M               44        20    harry     bar.c
       *        44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
A                0         ?     ?        stuff/things/bloo.h
Status against revision:   46
```

Observe los dos asteriscos: si usted ejecutara **svn update** en este punto, usted podría recibir cambios para `README` y `trout.c`. Esto le dice cierta información muy útilmdash;necesitará actualizar y coger los cambios del servidor para `README` antes de enviar sus cambios, o el repositorio rechazará su envío por estar fuera-de-fecha. (Más de este tema más adelante.)

svn diff

Otra manera de examinar sus cambios es con el comando **svn diff**. Puede descubrir *exactamente* cómo ha modificado cosas ejecutando **svn diff** sin argumentos, el cual imprime los cambios de los ficheros en formato unificado del diff:[5]

```
$ svn diff
Index: bar.c
===================================================================
--- bar.c (revision 3)
+++ bar.c (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
 return 0;
 }

Index: README
===================================================================
--- README (revision 3)
+++ README (working copy)
@@ -193,3 +193,4 @@
+Note to self:  pick up laundry.

Index: stuff/fish.c
===================================================================
```

[5] Subversion usa su motor interno de diff, el cual produce un formato unificado del diff, por defecto. Si quiere la salida del diff en un formato diferente, especifique un programa diff externo usando `--diff-cmd` y pasando cualquier parámetro que quiera usando la opción `--extensions`. Por ejemplo, para ver diferencias locales en el fichero `foo.c` en contexto con el formato de salida mientras ignora cambios en espacios en blanco puede ejecutar **svn diff --diff-cmd /usr/bin/diff --extensions '-bc' foo.c**.

```
--- stuff/fish.c (revision 1)
+++ stuff/fish.c (working copy)
-Welcome to the file known as 'fish'.
-Information on fish will be here soon.

Index: stuff/things/bloo.h
===================================================================
--- stuff/things/bloo.h (revision 8)
+++ stuff/things/bloo.h (working copy)
+Here is a new file to describe
+things about bloo.
```

El comando **svn diff** produce esta salida comparando sus ficheros de copia de trabajo contra la copia “prístina” almacenada en el área `.svn`. Los ficheros programados para la adición se visualizan como texto-añadido, y los ficheros programados para la eliminación son visualizados como texto eliminado.

La salida es visualizada en *formato unificado del diff*. Esto es, las lineas quitadas son empezadas con un `-` y las lineas añadidas son empezadas con un `+`. **svn diff** también imprime el nombre del fichero e información útil para el programa **patch**, así que puede generar “parches” redireccionando la salida del diff a un fichero:

```
$ svn diff > patchfile
```

Usted puede, por ejemplo, mandar por email el fichero de parche a otro desarrollador para la revisión o testeo antes de enviarlo.

svn revert

Ahora suponga que usted ve la salida del diff anterior, y se da cuenta que sus cambios a `README` son un error; quizás accidentalmente tecleó ese texto en el fichero equivocado en su editor

Esta es una oportunidad perfecta para usar **svn revert**.

```
$ svn revert README
Reverted 'README'
```

Subversion invierte el fichero a un estado pre-modificado reescribiéndolo con la copia “prístina” almacenada en el área `.svn`. Pero también observar que **svn revert** puede deshacer *cualquier* operación programada—por ejemplo, usted puede decidir que no quiere añadir un nuevo fichero después de todo:

```
$ svn status foo
?      foo

$ svn add foo
A         foo

$ svn revert foo
Reverted 'foo'

$ svn status foo
?      foo
```

svn revert *`ITEM`* tiene exactamente el mismo efecto que suprimiendo *`ITEM`* de su copia de trabajo local y después ejecutando **svn update -r BASE** *`ITEM`*. Sin embargo, si está revirtiendo un fichero, **svn revert** tiene una diferencia muy considerable—no tiene que comunicarse con el repositorio para reponer su fichero.

O quizás usted quito equivocadamente un fichero del control de versión:

```
$ svn status README
      README

$ svn delete README
D         README

$ svn revert README
Reverted 'README'

$ svn status README
      README
```

¡Mira mamá! ¡Sin red!

Estos tres comandos (**svn status**, **svn diff**, y **svn revert**) pueden ser usados sin ningún acceso de red. Esto lo hace sencillo para administrar sus cambios-en-progreso cuando usted está en alguna parte sin una conexión de red, tal como viajando en un avión, montando en un tren o hackeando en la playa.

Subversion hace esto manteniendo almacenes privados de versiones prístinas de cada fichero versionado dentro del área administrativa `.svn`. Esto permite a Subversion reportar—y revertir—modificaciones locales a esos ficheros *sin acceso de red*. Este almacén (llamado el "texto-base") también permite a Subversion mandar las modificaciones locales del usuario durante un envío al servidor como un *delta* comprimido (o "diferencial") contra la versión prístina. Tener este almacén es un beneficio tremendo—incluso si usted tiene una conexión de red rápida, es mucho más rápido mandar solamente los cambios del fichero al servidor antes que el fichero entero. A primera vista, esto puede no parecer tan importante, pero imagine la repercusión si usted intenta enviar un cambio de una línea a un fichero de 400MB ¡ y tiene que enviar el fichero entero al servidor!

Resolver conflictos (fusionando los cambios de otros)

Ya hemos visto cómo **svn status -u** puede predecir conflictos. Suponga que ejecuta **svn update** y ocurren algunas cosas interesantes

```
$ svn update
U  INSTALL
G  README
C  bar.c
Updated to revision 46.
```

Los códigos `U` y `G` no son causa para la inquietud; esos ficheros absorbieron limpiamente los cambios del repositorio. Los ficheros marcados con una `U` no contienen cambios locales pero fueron actUalizados con cambios del repositorio. La `G` representa merGed, lo que significa que el fichero tenía para comenzar cambios locales, pero los cambios que venían del repositorio no se solaparon de ninguna manera.

Pero la `C` representa conflicto. Esto significa que los cambios del servidor se solapan con los suyos propios, y ahora tiene que elegir manualmente entre ellos.

Siempre que ocurra un conflicto, ocurren tres cosas para ayudarle a notar y resolver ese conflicto:

- Subversion imprime una `C` durante la actualización, y recuerda que el fichero está en un estado de conflicto.

- Subversion coloca *marcas de conflicto*—secuencias especiales de texto que delimitan los "lados" del conflicto—en el fichero para demostrar visualmente las áreas solapadas.

- Para cada fichero en conflicto, Subversion coloca tres ficheros extra en su copia de trabajo local:

  `filename.mine`

  Este es su fichero como existió en su copia de trabajo antes de que usted actualizara su copia de trabajo—esto es, sin marcas de conflicto. Este fichero tiene su últimos cambios y nada más.

  `filename.rOLDREV`

  Este es el fichero que era la revisión `BASE` antes de que usted actualizará su copia de trabajo. Esto es, el fichero que usted descargó antes de que hiciera su última edición.

  `filename.rNEWREV`

  Este es el fichero que su cliente de Subversion recibió del servidor justo cuando usted actualizó su copia de trabajo. Este fichero corresponde con la revisión `HEAD` del repositorio.

  Aquí `OLDREV` es el número de revisión del fichero en su directorio `.svn` y `NEWREV` es el número de revisión del `HEAD` del repositorio.

Por ejemplo, Sally hace cambios al fichero `sandwich.txt` en el repositorio. Harry acaba de cambiar el fichero en su copia de trabajo y lo ha enviado. Sally actualiza su copia de trabajo antes de enviarlo y recibe un conflicto:

```
$ svn update
C  sandwich.txt
Updated to revision 2.
$ ls -1
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

En este punto, Subversion *no* le permitirá enviar el fichero `sandwich.txt` hasta que los tres ficheros temporales sean borrados.

```
$ svn commit --message "Add a few more things"
svn: Commit failed (details follow):
svn: Aborting commit: '/home/sally/svn-work/sandwich.txt' remains in conflict
```

Si obtiene un conflicto, necesita hacer una de tres cosas:

- Fusionar el texto en conflicto "a mano" (examinando y editando las marcas de conflicto dentro del fichero).

- Copiar uno de los ficheros temporales sobre su fichero de trabajo.

- Ejecutar **svn revert <filename>** para eliminar todos sus cambios locales.

Una vez que usted haya resuelto el conflicto, necesita dejar que Subversion lo sepa ejecutando **svn resolved**. Esto borrará los tres ficheros temporales y Subversion no considerará por más tiempo que el fichero está en estado de conflicto.[6]

```
$ svn resolved sandwich.txt
Resolved conflicted state of 'sandwich.txt'
```

Fusionando conflictos a mano

Fusionar conflictos a mano puede ser absolutamente intimidatorio la primera vez que lo haga, pero con un poco de practica, puede llegar a ser tan fácil como caerse de una bici.

Aquí tiene un ejemplo. Debido a una falta de comunicación, usted y Sally, su colaboradora, editan el fichero `sandwich.txt` en el mismo momento. Sally envía sus cambios, y cuando usted va a actualizar su copia de trabajo, obtiene un conflicto y vamos a tener que editar `sandwich.txt` para resolver los conflictos. Primero, echemos una hojeada al fichero:

```
$ cat sandwich.txt
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

Las líneas de signos menor-que, y signos mayor-que son marcas de conflictos, y no son parte de los datos en conflicto actuales. Generalmente usted querrá asegurarse que estén borrados del fichero antes de su próximo envío. El texto entre las dos primeras marcas están compuestas por los cambios que usted hizo en el área conflictiva:

```
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=======
```

El texto entre el segundo y tercer conjunto de marcas de conflicto es el texto del envío de Sally:

```
=======
Sauerkraut
Grilled Chicken
>>>>>>> .r2
```

Generalmente usted no deseará borrar las marcas de conflicto y los cambios de Sally—ella se sorprenderá terriblemente cuando llegue el sandwich y no sea lo que ella quería. Aquí es donde usted coge el teléfono o anda a través de la oficina y le explica a Sally que no puede tomar sauerkraut de un deli Italiano.[7] Una vez usted esté de acuerdo con los cambios deberá comprobarlos, editar su

[6] Siempre puede borrar los ficheros temporales usted mismo, pero ¿realmente querría hacer eso cuando Subversion puede hacerlo para usted? No lo creemos.

fichero y borrar las marcas de conflicto

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
Creole Mustard
Bottom piece of bread
```

Ahora ejecute **svn resolved**, y está listo para enviar sus cambios:

```
$ svn resolved sandwich.txt
$ svn commit -m "Go ahead and use my sandwich, discarding Sally's edits."
```

Recuerde, si alguna vez está confuso mientras edita el fichero conflictivo, siempre puede consultar los tres ficheros que Subversion crea para usted en su copia de trabajo—incluyendo su fichero como estaba antes de que usted actualizara. Incluso puede usar una herramienta interactiva de fusión de una third-party para examinar esos tres fichero.

Copiando un fichero en su fichero de trabajo

Si obtiene un conflicto y decide que quiere rechazar sus cambios, simplemente usted puede copiar uno de los ficheros temporales creados por Subversion sobre el fichero en su copia de trabajo:

```
$ svn update
C  sandwich.txt
Updated to revision 2.
$ ls sandwich.*
sandwich.txt  sandwich.txt.mine  sandwich.txt.r2  sandwich.txt.r1
$ cp sandwich.txt.r2 sandwich.txt
$ svn resolved sandwich.txt
```

Punting: Usando svn revert

Si obtiene un conflicto, y al examinarlo decide que quiere rechazar sus cambios y empezar su edición de nuevo, simplemente invierta sus cambios:

```
$ svn revert sandwich.txt
Reverted 'sandwich.txt'
$ ls sandwich.*
sandwich.txt
```

Observe que cuando usted invierte un fichero conflictivo, no tiene que ejecutar **svn resolved**.

Ahora usted está listo para enviar sus cambios. Observe que **svn resolved**, al contrario de la mayoría de los otros comandos que nos hemos ocupado en este capitulo, requiere un argumento. En cualquier caso, debe tener cuidado y solo ejecutar **svn resolved** cuando esté seguro que ha arreglado el conflicto en su fichero—una vez los ficheros temporales son borrados, Subversion le dejará enviar el fichero incluso si todavía tiene marcas de conflicto.

[7] Y si les pregunta por esto, pueden echarle de la ciudad en un carril.

Enviar sus cambios

¡Finalmente! Su edición está terminada, ha fusionado todos los cambios del servidor, y está listo para enviar sus cambios al repositorio.

El comando **svn commit** envía todos sus cambios al repositorio. Cuando usted envía un cambio, necesita proveer un *mensaje de registro*, describiendo su cambio. Su mensaje de registro será adjuntado a la nueva revisión que ha creado. Si su mensaje de registro es breve, puede querer proveerlo en la línea de comando usando la opción `--message` (o `-m`):

```
$ svn commit --message "Corrected number of cheese slices."
Sending        sandwich.txt
Transmitting file data .
Committed revision 3.
```

Sin embargo, si ha estado componiendo su mensaje de registro mientras trabaja, puede querer decirle a Subversion que coja el mensaje de un fichero pasando el nombre de fichero con la opción `--file` :

```
$ svn commit --file logmsg
Sending        sandwich
Transmitting file data .
Committed revision 4.
```

Si usted falla en especificar cualquiera de las opciones `--message` o `--file`, entonces Subversion lanzará automáticamente su editor favorito (según lo definido en la variable de entorno `$EDITOR`) para redactar un mensaje de registro.

Si está en su editor escribiendo un mensaje de registro y decide que quiere cancelar su envío, usted puede quitar su editor sin guardar los cambios. Si ya ha guardado su mensaje de registro, simplemente borre el texto y salve otra vez.

```
$ svn commit
Waiting for Emacs...Done

Log message unchanged or not specified
a)bort, c)ontinue, e)dit
a
$
```

El repositorio no sabe ni cuida si sus cambios tienen algún sentido en su totalidad; solo comprueba para asegurarse que nadie haya cambiado cualquiera de los mismos ficheros que usted mientras usted no miraba. Si alguien *ha* hecho esto, el envío entero fallará con un mensaje informándole que uno o más de sus ficheros está fuera-de-fecha:

```
$ svn commit --message "Add another rule"
Sending        rules.txt
svn: Commit failed (details follow):
svn: Out of date: 'rules.txt' in transaction 'g'
```

En este punto, necesita ejecutar **svn update**, ocupándose con cualquier fusión o conflicto que resulte y procure enviarlo otra vez.

Eso cubre el ciclo básico de trabajo para usar Subversion. Hay muchas otras características en Subversion que usted puede usar para administrar su repositorio y copia de trabajo, pero puede pasar fácilmente usando solo los comandos que hemos visto hasta ahora en este capítulo.

Examinando el historial

Como hemos mencionado anteriormente, el repositorio es como una máquina del tiempo. Este mantiene un expediente de cada cambio enviado, y le permite explorar este historial examinando versiones anteriores de ficheros y directorios así como los metadatos que los acompañan. Con un único comando de Subversion, puede descargar el repositorio (o restaurar una copia de trabajo existente) exactamente como era en cualquier fecha o número de revisión en el pasado. Sin embargo, a veces solo desea *mirar* al pasado en vez de *ir* al pasado.

Hay varios comandos que pueden proporcionarle datos históricos del repositorio:

svn log

Le muestra amplia información: mensajes de registro unidos a las revisiones, y que ruta de fichero cambió en cada revisión.

svn diff

Le muestra los detalles específicos de cómo cambió un fichero en un cierto plazo.

svn cat

Este se utiliza para recuperar cualquier fichero tal como existió en un un número de revisión particular y lo muestra en su pantalla.

svn list

Muestra los ficheros en un directorio para cualquier revisión dada.

svn log

Para descubrir información sobre la historia de un fichero o directorio, use el comando **svn log**. **svn log** le proporcionará un registro de quién hizo cambios a un fichero o directorio, en qué revisión cambió, la hora y fecha de esa revisión, y, si fue proporcionado, el mensaje de registro que acompañaba al envío.

```
$ svn log
------------------------------------------------------------------------
r3 | sally | Mon, 15 Jul 2002 18:03:46 -0500 | 1 line

Added include lines and corrected # of cheese slices.
------------------------------------------------------------------------
r2 | harry | Mon, 15 Jul 2002 17:47:57 -0500 | 1 line

Added main() methods.
------------------------------------------------------------------------
r1 | sally | Mon, 15 Jul 2002 17:40:08 -0500 | 1 line

Initial import
------------------------------------------------------------------------
```

Observe que los mensajes de registro son impresos en *orden cronológico inverso* por defecto. Si desea ver un rango diferente de revisiones en un orden particular, o solo una única revisión, pase la opción `--revision` (`-r`):

```
$ svn log --revision 5:19    # shows logs 5 through 19 in chronological order

$ svn log -r 19:5            # shows logs 5 through 19 in reverse order
```

```
$ svn log -r 8              # shows log for revision 8
```

También puede examinar el historial de registro de un único fichero o directorio. Por ejemplo:

```
$ svn log foo.c
…
$ svn log http://foo.com/svn/trunk/code/foo.c
…
```

Ésto mostrará los mensajes de registro *solo* para esas revisiones en las cuales el fichero de trabajo (o URL) cambió.

Si desea aún más información sobre un fichero o directorio, **svn log** también toma una opción `--verbose` (`-v`). Porque Subversion le permite mover y copiar ficheros y directorios, es importante poder seguir cambios de la ruta del fichero en el sistema de ficheros, así en modo detallado, **svn log** incluirá una lista de rutas de fichero cambiadas en una revisión en su salida:

```
$ svn log -r 8 -v
------------------------------------------------------------------------
r8 | sally | 2002-07-14 08:15:29 -0500 | 1 line
Changed paths:
M /trunk/code/foo.c
M /trunk/code/bar.h
A /trunk/code/doc/README

Frozzled the sub-space winch.

------------------------------------------------------------------------
```

¿Por qué svn log me da una respuesta vacía?

Después de trabajar con Subversion un poco, la mayoría de los usuarios tendrán algo como esto:

```
$ svn log -r 2
------------------------------------------------------------------------
$
```

A primer vistazo, esto parece como un error. Pero recuerde que mientras las revisiones son repository-wide, **svn log** funciona sobre una ruta en el repositorio. Si no provee ninguna ruta de fichero, Subversion usa el directorio de trabajo actual como destino por defecto. En consecuencia, si usted está trabajando en un subdirectorio de su copia de trabajo y procurando registrar una revisión en la cual ni ese directorio ni cualquiera de sus hijos fue cambiado, Subversion le dará un registro vacío. Si desea ver qué cambió en esa revisión, intente indicando **svn log** directamente en lo más alto del URL de su repositorio, como en **svn log -r 2 http://svn.collab.net/repos/svn**.

svn diff

Ya hemos visto **svn diff** antes—éste muestra las diferencias de fichero en un formato unificado del diff; fue utilizado para mostrar las modificaciones locales hechas a nuestra copia de trabajo antes de enviarlas al repositorio.

De hecho, resulta que hay *tres* usos distintos para **svn diff**:

- Examinar cambios locales

- Comparar su copia de trabajo con la del repositorio
- Comparar repositorio con repositorio

Examinando cambios locales

Como hemos visto, invocando **svn diff** sin argumentos comparará sus ficheros de trabajo con las copias "prístinas" almacenadas en el área `.svn`:

```
$ svn diff
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

Comparando copia de trabajo con repositorio

Si se pasa un único `--revision` (`-r`) número, entonces su copia de trabajo es comparada con la revisión especificada del repositorio.

```
$ svn diff --revision 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

Comparando repositorio con repositorio

Si dos números de revisión, separados por una coma, son pasados vía `--revision` (`-r`), entonces las dos revisiones son comparadas directamente.

```
$ svn diff --revision 2:3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
```

```
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth closed
$
```

No solo puede usar **svn diff** para comparar ficheros en su copia de trabajo con el repositorio, sino que si suministra una URL como argumento, usted puede examinar las diferencias entre elementos en el repositorio incluso sin tener una copia de trabajo. Esto es especialmente útil si desea inspeccionar cambios en un fichero cuando no tiene una copia de trabajo en su máquina local:

```
$ svn diff --revision 4:5 http://svn.red-bean.com/repos/example/trunk/text/rules.txt
…
$
```

svn cat

Si desea examinar una versión anterior de un fichero y no necesariamente las diferencias entre dos ficheros, puede usar **svn cat**:

```
$ svn cat --revision 2 rules.txt
Be kind to others
Freedom = Chocolate Ice Cream
Everything in moderation
Chew with your mouth closed
$
```

También puede redireccionar la salida directamente a un fichero:

```
$ svn cat --revision 2 rules.txt > rules.txt.v2
$
```

Probablemente se esté preguntando por qué no usamos **svn update --revision** para actualizar el fichero a la revisión más antigua. Hay algunas razones por las que preferimos usar **svn cat**.

Primero, usted puede querer ver las diferencias entre dos revisiones de un fichero usando un programa diff externo (quizás uno gráfico, o quizás su fichero está en un formato que la salida de un diff unificado es absurdo). En este caso, necesitará coger una copia de la revisión antigua, redireccionarla a un fichero, y pasar este y el fichero de su copia de trabajo a su programa diff externo.

A veces es más fácil mirar una versión más antigua de un fichero en su totalidad en comparación con las diferencias entre esta y otra revisión.

svn list

El comando **svn list** le muestra qué ficheros están en un directorio de un repositorio sin realmente descargar los ficheros a su máquina local:

```
$ svn list http://svn.collab.net/repos/svn
README
branches/
clients/
tags/
trunk/
```

Si desea un listado más detallado, pase la opción `--verbose` (`-v`) para obtener una salida como esta.

```
$ svn list --verbose http://svn.collab.net/repos/svn
   2755 harry          1331 Jul 28 02:07 README
   2773 sally               Jul 29 15:07 branches/
   2769 sally               Jul 29 12:07 clients/
   2698 harry               Jul 24 18:07 tags/
   2785 sally               Jul 29 19:07 trunk/
```

Las columnas le dicen la revisión en la cual el fichero o directorio fue modificado por última vez, el usuario qué lo modificó, el tamaño si este es un fichero, la fecha de la última modificación, y el nombre del objeto.

Una palabra final en el historial

Además de todos los comandos anteriores, usted puede usar **svn update** y **svn checkout** con la opción `--revision` para tomar una copia de trabajo entera "anterior en el tiempo" [8]:

```
$ svn checkout --revision 1729 # Checks out a new working copy at r1729
…
$ svn update --revision 1729 # Updates an existing working copy to r1729
…
```

Otros comandos útiles

Mientras que no son usados con tanta frecuencia como los comandos discutidos previamente en este capítulo, usted necesitará de vez en cuando estos comandos.

svn cleanup

Cuando Subversion modifica su copia de trabajo (o cualquier información en el interior de `.svn`), intenta hacerlo tan seguro como sea posible. Antes de cambiar cualquier cosa, escribe sus intenciones en un fichero de registro, ejecuta los comandos en el fichero de registro, entonces borra el fichero de registro (esto es similar en diseño a un sistema de ficheros transaccional). Si una operación de Subversion es interrumpida (si el proceso es matado, o si la máquina se cuelga, por ejemplo), los ficheros de registro permanecen en disco. Ejecutando de nuevo los ficheros de registro, Subversion puede completar la operación anteriormente empezada, y su copia de trabajo puede volver a estar en un estado consistente.

Y esto es exactamente lo que hace **svn cleanup**: busca en su copia de trabajo y ejecuta cualquier registro de sobra, eliminando bloqueos en el proceso. Si Subversion alguna vez le dice que alguna parte de su copia de trabajo está "bloqueada", entonces éste es el comando que debería ejecutar. También, **svn status** mostrará una `L` al lado de los objetos bloqueados:

```
$ svn status
  L    somedir
M      somedir/foo.c

$ svn cleanup
$ svn status
M      somedir/foo.c
```

[8] ¿Ve? Le dijimos que Subversion era una máquina del tiempo.

svn import

El comando **svn import** es una manera rápida de copiar un árbol de ficheros sin versionar en el repositorio, creando directorios intermedios como sea necesario.

```
$ svnadmin create /usr/local/svn/newrepos
$ svn import mytree file:///usr/local/svn/newrepos/some/project
Adding         mytree/foo.c
Adding         mytree/bar.c
Adding         mytree/subdir
Adding         mytree/subdir/quux.h

Committed revision 1.
```

El ejemplo anterior copia el contenido del directorio `mytree` debajo del directorio `some/project` en el repositorio:

```
$ svn ls file:///usr/local/svn/newrepos/some/project
bar.c
foo.c
subdir/
```

Observe que después de que la importación esté acabada, el árbol original *no* se convierte en una copia de trabajo. Para empezar a trabajar, usted todavía necesita hacer **svn checkout** en una copia de trabajo fresca del árbol.

Sumario

Ahora hemos cubierto la mayoría de los comandos del cliente de Subversion. Las excepciones notables son ésas que se ocupan de la ramificación y de la combinación (vea Capítulo 4, *Crear ramas y fusionarlas*) y de las propiedades (vea "Propiedades"). Sin embargo, usted puede querer tomarse un momento para echar un ojo a Capítulo 9, *Referencia completa de Subversion* para tener una idea de todos los muchos comandos diferentes que tiene Subversion—y cómo puede usarlos para hacer su trabajo más fácil.

Capítulo 4. Crear ramas y fusionarlas

Crear ramas, etiquetar y fusionar, son conceptos comunes en casi todos los sistemas de control de versiones. Si no está familiarizado con estas ideas, le proporcionaremos una buena introducción en este capítulo. Si está familiarizado, esperamos que encuentre interesante descubrir cómo Subversion implementa estas ideas.

Crear ramas es una parte fundamental del control de versiones. Si va a permitir que Subversion gestione sus datos, entonces esta es una característica de la cual acabará dependiendo. Este capítulo asume que ya está familiarizado con los conceptos básicos de Subversion (Capítulo 2, *Conceptos básicos*).

¿Qué es una rama?

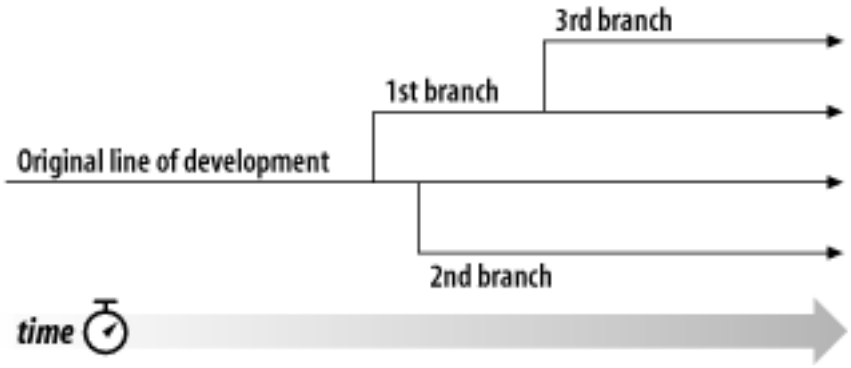
Supongamos que su trabajo es mantener un documento para una de las divisiones de su compañía, una especie de manual. Un día una división diferente le pide ese manual, pero con algunas partes 'retocadas' para ellos, dado que hacen las cosas de forma ligeramente diferente.

¿Qué hace en esta situación? Lo más obvio: realiza una segunda copia de su documento, y comienza a mantener ambas copias de forma separada. A medida que cada departamento solicita pequeños cambios, usted los incorpora en una de las copias o la otra.

A menudo desea realizar el mismo cambio en ambas copias. Por ejemplo, si descubre un error ortográfico en la primera copia, es muy probable que el mismo error exista en la segunda copia. Al fin y al cabo, ambos documentos son casi iguales; sólo se diferencian en pequeños detalles específicos.

Este es el concepto de una *rama*— una línea de desarrollo que existe de forma independiente a otra, pero comparte una historia común si mira suficientemente atrás en el tiempo. Una rama siempre nace como una copia de algo, y a partir de ahí, pasa a generar su propia historia (vea Figura 4.1, “Ramas de desarrollo”).

Figura 4.1. Ramas de desarrollo



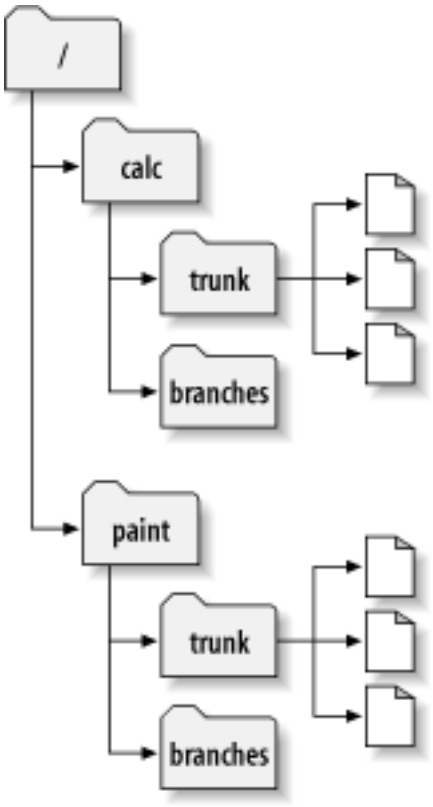
Subversion tiene comandos para ayudarle a mantener ramas paralelas de sus ficheros y directorios. Le permite crear ramas copiando sus datos, y recordando que las copias están relacionadas unas a otras. También le ayuda a duplicar cambios de una rama a otra. Finalmente, puede reflejar el contenido de diferentes ramas en partes de su copia de trabajo local, para que pueda “mezclar y probar” diferentes líneas de desarrollo en su trabajo diario.

Usando ramas

A estas alturas debería entender que todo cambio en el repositorio genera internamente un nuevo árbol de sistema de ficheros (llamado “revisión”). Si no es así, debería volver atrás y leer información sobre revisiones en “Revisiones”.

En este capítulo, volveremos al ejemplo del segundo capítulo. Recuerde cómo usted y su compañera, Carmen, comparten un repositorio que contiene dos proyectos, `paint` y `calc`. Fíjese que en Figura 4.2, “Estructura inicial del repositorio”, no obstante, cada directorio de proyecto contiene subdirectorios llamados `trunk` y `branches`. La razón se descubrirá pronto.

Figura 4.2. Estructura inicial del repositorio



Igual que antes, asuma que tanto Carmen como usted tienen copias locales del proyecto “calc”. En concreto, ambos tienen una copia local de `/calc/trunk`. Todos los ficheros del proyecto están en este subdirectorio en lugar de `/calc`, porque su equipo ha decidido que `/calc/trunk` es donde tendrá lugar la “línea principal” de desarrollo.

Digamos que le han encargado la tarea de realizar una reorganización radical del proyecto. Le tomará un largo tiempo realizar la modificación, y ésta afectará a todos los ficheros del proyecto. En esta situación el problema es que no quiere interferir con Carmen, quien sigue en el proceso de corregir pequeños fallos aquí y allá. Ella depende del hecho que la última versión del proyecto (en `/calc/trunk`) siempre sea usable. Si comienza a enviar cambios al repositorio poco a poco, seguramente fastidiará la tarea de Carmen.

Una estrategia es esconderse en un agujero: usted y Carmen pueden dejar de compartir información durante una o dos semanas. Es decir, comienza a reorganizar los ficheros en su copia local, pero no envía los cambios al servidor o actualiza su copia hasta que ha completado la tarea. Sin embargo, hay varios problemas con esto. En primer lugar, no es muy seguro. La mayoría de las personas prefieren guardar sus cambios en el repositorio con frecuencia, por si algo malo pudiera suceder de forma accidental a su copia local. Segundo, no es muy flexible. Si realiza su trabajo en diferentes ordenadores (quizás tiene una copia de `/calc/trunk` en dos máquinas diferentes), necesitará copiar manualmente sus cambios de un lado a otro, o realizar todo el trabajo en un solo ordenador. Del mismo modo, es difícil compartir sus cambios a mitad del desarrollo con otras personas. Una “práctica ideal” común en el desarrollo de software es permitir que sus compañeros puedan revisar su trabajo a medida que progresa. Si nadie ve los cambios que realiza poco a poco, pierde críticas potenciales. Finalmente, si ha finalizado con su tarea, puede encontrarse con que es muy difícil fusionar su trabajo final con el cuerpo principal del repositorio usado por el resto de la compañía. Carmen (y otros) puede haber realizado cambios en el repositorio que son difíciles de incorporar en su copia local—especialmente si ejecuta **svn update** semanas

tras el aislamiento.

La mejor solución es crear su propia rama, o línea de desarrollo, en el repositorio. Esto le permite guardar su trabajo a medio hacer con frecuencia sin interferir con otros, permitiendo a su vez compartir de forma selectiva información con sus colaboradores. Más adelante verá exactamente como funciona esto.

Creando una rama

Crear una rama es muy simple—realice una copia del proyecto en el repositorio usando el comando **svn copy**. Subversion no sólo es capaz de copiar ficheros individuales, sino también directorios. En este caso, desea realizar una copia del directorio `/calc/trunk`. ¿Dónde debería colocar la nueva copia? Donde desee—es una cuestión de política de su proyecto. Digamos que su equipo tiene la política de crear ramas en el área de repositorio `/calc/branches`, y quiere nombrar la rama `my-calc-branch`. Entonces querrá crear un nuevo directorio `/calc/branches/my-calc-branch`, el cual comienza su vida como una copia de `/calc/trunk`.

Hay dos formas diferentes de realizar una copia. Le mostraremos primero el modo engorroso, para asegurarnos de que queda claro el concepto. Para comenzar, obtenga una copia local del directorio raíz del proyecto, `/calc`:

```
$ svn checkout http://svn.example.com/repos/calc bigwc
A  bigwc/trunk/
A  bigwc/trunk/Makefile
A  bigwc/trunk/integer.c
A  bigwc/trunk/button.c
A  bigwc/branches/
Checked out revision 340.
```

Realizar ahora una copia es cuestión de pasar dos rutas locales al comando **svn copy**:

```
$ cd bigwc
$ svn copy trunk branches/my-calc-branch
$ svn status
A  +   branches/my-calc-branch
```

En este caso, el comando **svn copy** copia recursivamente el directorio `trunk` en un nuevo directorio, `branches/my-calc-branch`. Tal y como puede ver por el comando **svn status**, el nuevo directorio está ahora a la espera de ser añadido al repositorio. Pero fíjese en el signo “+” tras la letra A. Esto indica que la adición pendiente es una *copia* de algo, no algo nuevo. Cuando envíe sus cambios al repositorio, Subversion creará `/calc/branches/my-calc-branch` en el repositorio como una copia de `/calc/trunk`, en lugar de reenviar todos los datos de su copia local de nuevo a través de la red:

```
$ svn commit -m "Creating a private branch of /calc/trunk."
Adding         branches/my-calc-branch
Committed revision 341.
```

Y ahora el método más sencillo para crear una rama, que deberíamos haberle explicado en primer lugar: **svn copy** es capaz de operar directamente sobre dos URLs.

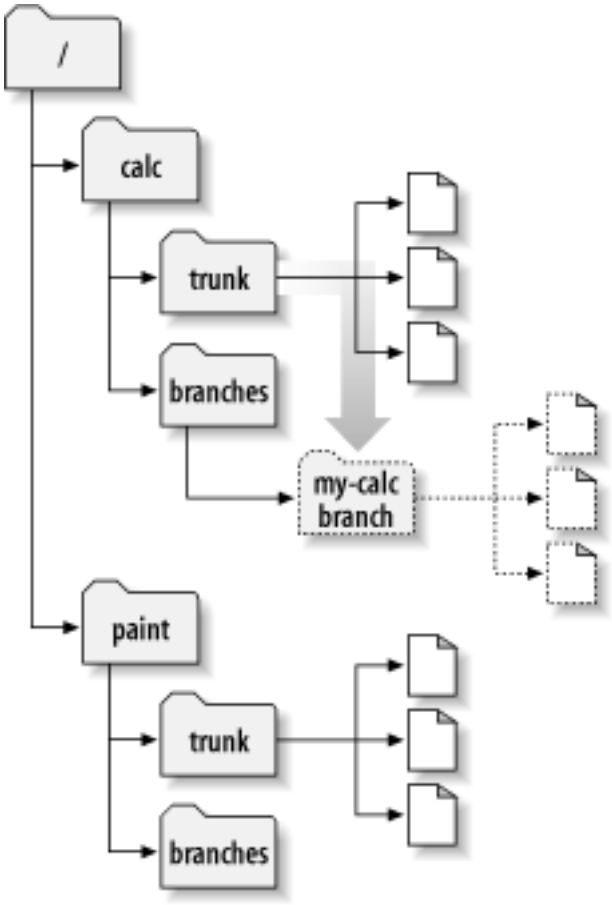
```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Creating a private branch of /calc/trunk."

Committed revision 341.
```

Realmente no hay diferencia entre estos dos métodos. Ambos procedimientos crean un nuevo directorio en la revisión 341, y el

nuevo directorio es una copia de `/calc/trunk`. Esto se puede ver en Figura 4.3, “Repositorio con nueva copia”. Note cómo el segundo método, no obstante, realiza los cambios en el repositorio *inmediatamente*. [1] Es un procedimiento más sencillo, porque no requiere obtener primero una gran copia local de todo el repositorio. De hecho, esta técnica ni si quiera requiere que tenga una copia local en absoluto.

Figura 4.3. Repositorio con nueva copia



Copias ligeras

El repositorio de Subversion sigue un diseño especial. Cuando copia un directorio, no necesita preocuparse por el crecimiento del repositorio—en realidad Subversion no duplica los datos. En su lugar, crear una nueva entrada de directorio que apunta a un árbol *existente*. Si es un usuario de Unix, este es el mismo concepto que un enlace duro. A partir de ahí, se dice que la copia es “vaga”. Es decir, si realiza un cambio a un fichero contenido en el directorio copiado, sólo cambia ese fichero—el resto de los ficheros continúan su existencia como enlaces a los ficheros originales del directorio original.

Por esta razón escuchará a los usuarios de Subversion hablar con frecuencia sobre “copias ligeras”. No importa cuan grande sea el directorio—sólo requiere un tiempo constante muy pequeño realizar una copia del mismo. De hecho, esta es la característica en la que se basan los cambios realizados en Subversion: cada revisión es una “copia ligera” de la revisión anterior, con algunos elementos cambiados. (Para aprender más sobre esto, visite la página web de Subversion y lea en los documentos sobre el diseño de Subversion el método “bubble up”.)

[1] Subversion no soporta copias entre diferentes repositorios. Cuando use URLs con **svn copy** o **svn move**, sólo puede copiar elementos dentro del mismo repositorio.

Por supuesto, este mecanismo interno usado para copiar y compartir datos está oculto al usuario, que simplemente ve copias de árboles de ficheros. Lo importante es saber que las copias son ligeras, tanto en tiempo como espacio. Cree tantas ramas como desee.

Trabajando con su rama

Ahora que ha creado una rama del proyecto, puede obtener una nueva copia local para comenzar a usarla:

```
$ svn checkout http://svn.example.com/repos/calc/branches/my-calc-branch
A  my-calc-branch/Makefile
A  my-calc-branch/integer.c
A  my-calc-branch/button.c
Checked out revision 341.
```

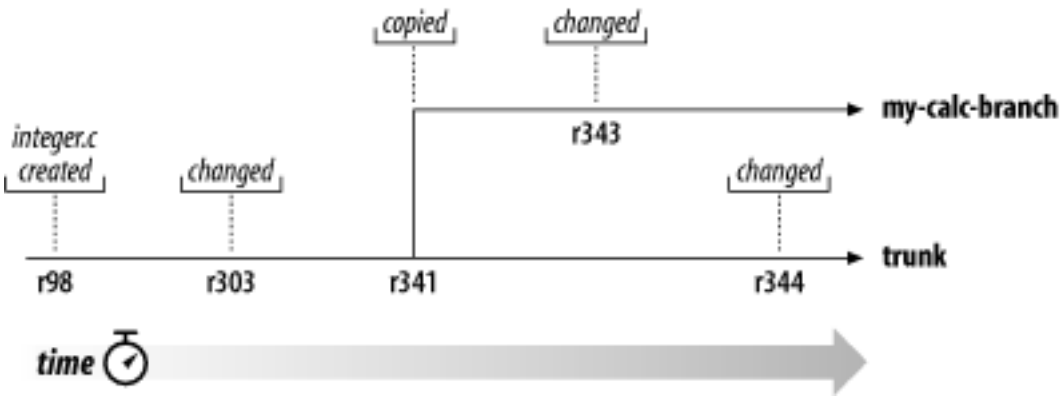
No hay nada especial sobre esta copia local; simplemente refleja un directorio diferente del repositorio. Cuando envíe sus cambios, no obstante, Carmen nunca los verá cuando se actualice. Su copia local es de `/calc/trunk`. (Asegúrese de leer "Cambiando la copia local de trabajo" más adelante en este capítulo: el comando **svn switch** permite crear una copia local de una rama de modo diferente.)

Pretendamos que ha pasado una semana, y se realizan los siguientes cambios:

- Ha realizado un cambio en `/calc/branches/my-calc-branch/button.c`, creando la revisión 342.
- Ha realizado un cambio en `/calc/branches/my-calc-branch/integer.c`, creando la revisión 343.
- Carmen ha realizado un cambio en `/calc/trunk/integer.c`, creando la revisión 344.

Ahora hay dos líneas independientes de desarrollo, mostradas en Figura 4.4, "Bifurcación de la historia de un fichero", sobre el fichero `integer.c`.

Figura 4.4. Bifurcación de la historia de un fichero



Las cosas se ponen interesantes cuando mira el historial de cambios realizados a su copia de `integer.c`:

```
$ pwd
/home/usuario/my-calc-branch

$ svn log --verbose integer.c
------------------------------------------------------------------------
r343 | usuario | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   M /calc/branches/my-calc-branch/integer.c

* integer.c:  frozzled the wazjub.

------------------------------------------------------------------------
r341 | usuario | 2002-11-03 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   A /calc/branches/my-calc-branch (from /calc/trunk:340)

Creating a private branch of /calc/trunk.

------------------------------------------------------------------------
r303 | carmen | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  changed a docstring.

------------------------------------------------------------------------
r98 | carmen | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  adding this file to the project.

------------------------------------------------------------------------
```

Fíjese como Subversion sigue la historia de `integer.c` en su rama hacia atrás, traspasando incluso el punto donde el fichero fue copiado. Muestra la creación de la rama como un evento en la historia, porque `integer.c` fue copiado de forma implícita cuando realizó la copia de `/calc/trunk/`. Ahora mire qué es lo que ocurre cuando Carmen ejecuta el mismo comando sobre su copia del fichero:

```
$ pwd
/home/carmen/calc

$ svn log --verbose integer.c
------------------------------------------------------------------------
r344 | carmen | 2002-11-07 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  fix a bunch of spelling errors.

------------------------------------------------------------------------
r303 | carmen | 2002-10-29 21:14:35 -0600 (Tue, 29 Oct 2002) | 2 lines
Changed paths:
   M /calc/trunk/integer.c

* integer.c:  changed a docstring.

------------------------------------------------------------------------
r98 | carmen | 2002-02-22 15:35:29 -0600 (Fri, 22 Feb 2002) | 2 lines
Changed paths:
```

```
   M /calc/trunk/integer.c

* integer.c:  adding this file to the project.

------------------------------------------------------------------------
```

Carmen ve su propio cambio de la revisión 344, pero no el cambio realizado en la revisión 343. Desde el punto de vista de Subversion, estos dos cambios afectaron ficheros diferentes en distintos lugares del repositorio. No obstante, Subversion *muestra* que ambos ficheros comparten una historia común. Antes de que la rama fuese creada en la revisión 341, ambos eran el mismo fichero. Por esta razón tanto usted como Carmen ven los cambios realizados en las revisiones 303 y 98.

Conceptos clave sobre las ramas

Hay dos lecciones importantes que debería recordar de esta sección.

1. A diferencia de muchos otros sistemas de control de versiones, las ramas de Subversion existen como *directorios normales del sistema de archivos* en el repositorio, no en una dimensión extra. Estos directorios simplemente llevan información histórica adicional.

2. Subversion no tiene un concepto interno de rama—sólo copias. Cuando copia un directorio, el nuevo directorio sólo es una "rama" porque *usted* añade esa connotación. Puede considerar el directorio de forma diferente, o tratarlo de manera diferente, pero para Subversion no es más que un directorio ordinario que simplemente fue creado como resultado de una operación de copiado.

Copiando cambios entre ramas

Ahora usted y Carmen están trabajando en ramas paralelas del proyecto: usted está trabajando en una rama privada, y Carmen está trabajando en el *tronco*, o línea principal de desarrollo.

En proyectos que tienen grandes cantidades de contribuyentes, es habitual que la mayoría tengan copias locales del tronco. Cuando alguien necesita hacer un cambio a largo plazo que puede molestar en el tronco, un procedimiento estándar es crear una rama privada y realizar ahí los cambios hasta que todo el trabajo esté completo.

Así que las buenas noticias son que usted y Carmen no están interfiriendo en sus trabajos. Las malas noticias son que es muy fácil derivar *demasiado* lejos. Recuerde que uno de los problemas con la estrategia "esconderse en un agujero" es que para cuando haya terminado con su rama, puede que sea casi imposible fusionar los cambios con el tronco sin un alto número de conflictos.

En su lugar, usted y Carmen pueden continuar compartiendo cambios a medida que trabajan. Es tarea suya decidir qué cambios merece la pena compartir; Subversion le brinda la posibilidad de "copiar" cambios entre ramas de forma selectiva. Y para cuando haya finalizado completamente con su rama, su conjunto entero de cambios en la rama puede ser copiado en el tronco.

Copiando cambios específicos

El la sección anterior, mencionamos que tanto usted como Carmen han realizado cambios a `integer.c` en ramas diferentes. Si observa el informe de cambios de la revisión 344, puede ver que ella realizó algunas correcciones de ortografía. Sin duda, su copia del mismo fichero debe tener todavía los mismos errores de ortografía. Es probable que futuros cambios al fichero afectarán las mismas áreas que tienen estos errores ortográficos, lo que potencialmente generará conflictos cuando fusione su rama algún día. Es mejor, entonces, recibir ahora los cambios de Carmen, *antes* de que comience a trabajar de forma intensiva en el mismo sitio.

Es el momento de usar el comando **svn merge**. Este comando, descubrirá que es un primo cercano del comando **svn diff** (sobre el cual leímos en el tercer capítulo). Ambos comandos son capaces de comparar dos objetos cualquiera del repositorio y describir sus diferencias. Por ejemplo, puede preguntar a **svn diff** que le muestre el cambio exacto realizado por Carmen en la revisión 344:

```
$ svn diff -r 343:344 http://svn.example.com/repos/calc/trunk

Index: integer.c
===================================================================
--- integer.c (revision 343)
+++ integer.c (revision 344)
@@ -147,7 +147,7 @@
     case 6:  sprintf(info->operating_system, "HPFS (OS/2 or NT)"); break;
     case 7:  sprintf(info->operating_system, "Macintosh"); break;
     case 8:  sprintf(info->operating_system, "Z-System"); break;
-    case 9:  sprintf(info->operating_system, "CPM"); break;
+    case 9:  sprintf(info->operating_system, "CP/M"); break;
     case 10:  sprintf(info->operating_system, "TOPS-20"); break;
     case 11:  sprintf(info->operating_system, "NTFS (Windows NT)"); break;
     case 12:  sprintf(info->operating_system, "QDOS"); break;
@@ -164,7 +164,7 @@
     low = (unsigned short) read_byte(gzfile);  /* read LSB */
     high = (unsigned short) read_byte(gzfile); /* read MSB */
     high = high << 8;  /* interpret MSB correctly */
-    total = low + high; /* add them togethe for correct total */
+    total = low + high; /* add them together for correct total */

     info->extra_header = (unsigned char *) my_malloc(total);
     fread(info->extra_header, total, 1, gzfile);
@@ -241,7 +241,7 @@
      Store the offset with ftell() ! */

   if ((info->data_offset = ftell(gzfile))== -1) {
-    printf("error: ftell() retturned -1.\n");
+    printf("error: ftell() returned -1.\n");
     exit(1);
   }

@@ -249,7 +249,7 @@
   printf("I believe start of compressed data is %u\n", info->data_offset);
   #endif

-  /* Set postion eight bytes from the end of the file. */
+  /* Set position eight bytes from the end of the file. */

   if (fseek(gzfile, -8, SEEK_END)) {
     printf("error: fseek() returned non-zero\n");
```

El comando **svn merge** es casi exactamente idéntico. En lugar de mostrar las diferencias en su terminal, no obstante, las aplica directamente a su copia local como *modificaciones locales*:

```
$ svn merge -r 343:344 http://svn.example.com/repos/calc/trunk
U  integer.c

$ svn status
M  integer.c
```

La salida del comando **svn merge** muestra que su copia de `integer.c` fue parcheada. Ahora contiene el cambio de Carmen—el cambio ha sido "copiado" desde el tronco a su copia local de la rama privada, y existe ahora como una modificación local. En este punto, es tarea suya revisar la modificación local para asegurarse de que funciona correctamente.

En otra situación, es posible que las cosas no hayan ido tan bien, y que `integer.c` haya entrado en un estado de conflicto. Puede tener que resolver el conflicto usando el procedimiento estándar (vea el tercer capítulo), o si decide que la fusión fue una mala idea, simplemente dé se por vencido y use **svn revert** para revertir el cambio local.

Pero asumiendo que ha revisado el cambio fusionado, puede guardar sus cambios como es habitual con **svn commit**. En ese punto, el cambio ha sido fusionado en su rama del repositorio. En terminología de control de versiones, este acto de copiar cambios entre ramas se denomina *portar* cambios.

Cuando guarda su modificación local, asegúrese de que el mensaje de cambios indica que está portando un cambio específico de una rama a otra. Por ejemplo:

```
$ svn commit -m "integer.c: ported r344 (spelling fixes) from trunk."
Sending        integer.c
Transmitting file data .
Committed revision 360.
```

Tal y como verá en las siguientes secciones, es muy importante seguir esta “práctica recomendada”.

¿Por qué no usar parches?

Una pregunta que tendrá en mente, especialmente si es un usuario de Unix: ¿para qué usar el comando **svn merge**? ¿Por qué no simplemente usar el comando **patch** del sistema operativo para realizar la misma tarea? Por ejemplo:

```
$ svn diff -r 343:344 http://svn.example.com/repos/calc/trunk > patchfile
$ patch -p0  < patchfile
Patching file integer.c using Plan A...
Hunk #1 succeeded at 147.
Hunk #2 succeeded at 164.
Hunk #3 succeeded at 241.
Hunk #4 succeeded at 249.
done
```

En este caso particular, efectivamente, no hay realmente diferencia alguna. Pero **svn merge** tiene características especiales que sobrepasan las del programa **patch**. El formato de fichero usado por **patch** es bastante limitado; sólo permite modificar el contenido de los ficheros. No hay forma de representar cambios a *árboles*, como añadir, eliminar o renombrar ficheros y directorios. Si el cambio de Carmen, digamos, hubiese añadido un nuevo directorio, la salida de **svn diff** ni si quiera lo hubiese mencionado. **svn diff** sólo genera salida en el formato limitado de patch, por lo que hay algunas ideas que no puede expresar. [2] El comando **svn merge**, no obstante, puede expresar cambios a árboles directamente aplicándolos a su copia local.

Aviso: aunque los comandos **svn diff** y **svn merge** son similares en concepto, tienen una diferente sintaxis en muchos casos. Asegúrese de leer en el capítulo 8 los detalles, o pregunte a **svn help**. Por ejemplo, **svn merge** requiere una ruta a una copia local como destino, es decir, un lugar donde debe aplicar los cambios del árbol. Si el destino no es especificado, asume que está intentando realizar una de las siguientes operaciones comunes:

1. Quiere fusionar cambios de directorio directamente en su copia local.

2. Quiere fusionar los cambios de un fichero específico en un fichero que existe con el mismo nombre en su directorio de trabajo actual.

Si está fusionando un directorio y no ha especificado una ruta destino, **svn merge** asume el primer caso mencionado e intenta aplicar los cambios en su directorio actual. Si está fusionando un fichero, y ese fichero (o un fichero con el mismo nombre) existe en su directorio actual de trabajo, **svn merge** asume el segundo caso mencionado e intenta aplicar los cambios al fichero local de mismo nombre.

[2] En el futuro, el proyecto Subversion planea usar (o inventar) un formato de parche que describe cambios a árboles.

Si quiere aplicar los cambios en otro lugar, tendrá que especificarlo. Por ejemplo, si todavía está en el directorio padre de su copia local, tendrá que especificar el directorio destino para recibir los cambios:

```
$ svn merge -r 343:344 http://svn.example.com/repos/calc/trunk my-calc-branch
U   my-calc-branch/integer.c
```

Procedimientos ideales de fusionado

Realizar fusiones de forma manual

Fusionar cambios suena bastante simple, pero en la práctica puede convertirse en un dolor de cabeza. El problema es que si fusiona cambios con frecuencia de una rama a otra, puede acabar fusionando accidentalmente el mismo cambio *dos veces*. Cuando esto ocurre, a veces las cosas seguirán funcionando bien. Cuando aplica un parche a un fichero, Subversion normalmente se da cuenta de que el fichero ya contiene el cambio, y no hace nada. Pero si el cambio ya existente fue modificado de algún modo, obtendrá un conflicto.

Idealmente, su sistema de control de versiones debería prevenir la doble aplicación de cambios a una rama. Debería recordar automáticamente qué cambios fueron recibidos en la rama, y ser capaz de mostrarle un listado de los mismos. Debería poder usar esta información para automatizar las fusiones tanto como sea posible.

Desafortunadamente, Subversion no es tal sistema. Al igual que CVS, Subversion 1.0 no almacena ninguna información sobre operaciones de fusionado. Cuando envía sus cambios locales, el repositorio no tiene idea si esos cambios vinieron de una ejecución del comando **svn merge**, o de una modificación manual de los ficheros.

¿Qué significa esto para usted, el usuario? Significa que hasta que llegue el día que Subversion desarrolle esta característica, tendrá que gestionar usted mismo la información de fusionado. El mejor lugar para hacerlo es en el informe de cambios cuando envía cambios al repositorio. Tal y como se demostró en el ejemplo anterior, es recomendable que su informe de cambios mencione el número de revisión específico (o rango de revisiones) que está siendo fusionado en su rama. Más tarde, puede ejecutar **svn log** para revisar qué cambios contiene ya su rama. Esto le permitirá construir con cuidado siguientes comandos **svn merge** que no serán redundantes con cambios previamente portados.

En la siguiente sección, le mostraremos algunos ejemplos de esta técnica en acción.

Visualización previa de fusiones

Dado que el fusionado sólo genera modificaciones locales, normalmente no es una operación de alto riesgo. Si se equivoca en el fusionado la primera vez, simplemente ejecute **svn revert** para ignorar los cambios y pruebe de nuevo.

Es posible, no obstante, que su copia local ya tenga algunas modificaciones. Los cambios aplicados por una fusión se mezclarán con sus modificaciones locales, por lo que ejecutar **svn revert** ya no es una opción. Los dos conjuntos de cambios quizás sean imposibles de separar.

En casos como este, la gente se reconforta con la idea de ser capaces de predecir o examinar las fusiones antes de que ocurran. Una manera simple de hacerlo es ejecutar **svn diff** con los mismos argumentos que planea pasar a **svn merge**, tal y como ya le enseñamos en el primer ejemplo de fusionado. Otro método de visualización previa es pasar la opción `--dry-run` al comando de fusionado:

```
$ svn merge --dry-run -r 343:344 http://svn.example.com/repos/calc/trunk
U  integer.c

$ svn status
#  nothing printed, working copy is still unchanged.
```

La opción `--dry-run` en realidad no aplica ningún cambio a su copia local. Únicamente muestra los códigos de estado que *se-*

rían mostrados durante una fusión de verdad. Esto es útil para obtener una visualización previa de "alto nivel" de la potencial fusión, para aquellos casos en los que ejecutar **svn diff** proporcionaría demasiados detalles.

Subversion y los changesets

Todo el mundo parece tener una idea diferente de la definición de "changeset", o al menos expectaciones diferentes sobre lo que significa para un sistema de control de versiones tener "características de changeset". Para nuestros propósitos, digamos que un changeset es una colección de cambios con nombre único. Los cambios pueden incluir ediciones textuales a los contenidos de un fichero, modificaciones a la estructura de un directorio, o cambios en los meta datos. En el habla habitual, un changeset es un parche con un nombre al que se puede referir.

En Subversion, un número global de revisión N nombra el árbol en el repositorio: es la forma en la que quedó el repositorio tras el cambio número N. También es el nombre implícito de un changeset: si compara el árbol N con el árbol N-1, puede derivar el parche exacto cuyos cambios fueron hechos efectivos. Por esta razón, es fácil pensar de la "revisión N" no sólo como un árbol, sino también como un changeset. Si usa un software de gestión de tareas para tratar fallos, puede usar los números de revisión para hacer referencia a los parches específicos que corrigen determinados fallos—por ejemplo, "este problema fue corregido en la revisión 9238.". Entonces alguien puede ejecutar **svn log -r9238** para obtener información del changeset exacto que corrigió el problema, y ejecutar **svn diff -r9237:9238** para ver el propio parche. Y el comando de fusionado de Subversion también usa números de revisión. Puede fusionar changesets específicos de una rama a otra nombrándolos en los parámetros de fusionado: **svn merge -r9237:9238** fusionaría el changeset #9238 en su copia local.

Teniendo en cuenta o ignorando ascendencia

Cuando hable con un desarrollador de Subversion, es muy posible que oiga alguna referencia al término *ascendencia*. Esta palabra es usada para describir la relación entre dos objetos en un repositorio: si ambos están relacionados, entonces se dice que un objeto es el ancestro del otro.

Por ejemplo, suponga que envía cambios al repositorio en la revisión 100, los cuales incluyen un cambio al fichero `foo.c`. Entonces `foo.c@99` es un "ancestro" de `foo.c@100`. Por otro lado, suponga que hace efectivo el borrado de `foo.c` en la revisión 101, y entonces añade un nuevo fichero con el mismo nombre en la revisión 102. En este caso, `foo.c@99` y `foo.c@102` pueden parecer relacionados (tienen la misma ruta), pero de hecho son objetos completamente diferentes en el repositorio. No comparten historial o "ascendencia".

La razón de mencionar esto ahora es para señalar una diferencia importante entre **svn diff** y **svn merge**. El primer comando ignora la ascendencia, mientras que el segundo es muy sensitivo a ella. Por ejemplo, si usase **svn diff** para comparar las revisiones 99 y 102 de `foo.c`, vería ficheros diferenciales basados en líneas; el comando diff esta comparando a ciegas ambas rutas. Pero si usase **svn merge** para comparar ambos objetos, se daría cuenta de que no están relacionados y primero intentaría borrar el fichero antiguo, y entonces añadir el nuevo; vería `D foo.c` seguido por `A foo.c`.

En la mayoría de las fusiones se comparan árboles que están relacionados por ascendencia, y por esta razón **svn merge** sigue este comportamiento por defecto. No obstante, ocasionalmente puede querer que el comando de fusionado compare dos árboles no relacionados. Por ejemplo, puede haber importado dos árboles de código fuente representando diferentes versiones de un proyecto de software (lea "Ramas de proveedores"). ¡Si usase **svn merge** para comparar los dos árboles, vería que el primero es borrado completamente, seguido de una adición completa del segundo!

En estas situaciones, querrá que **svn merge** realice una comparación basada únicamente en rutas de fichero, ignorando cualquier relación existente entre ficheros y directorios. Añada la opción `--ignore-ancestry` a su comando de fusionado, y se comportará del mismo modo que **svn diff**. (Y de modo similar, la opción `--notice-ancestry` hará que **svn diff** se comporte como un comando de fusionado.)

Casos habituales de fusionado

Hay muchos usos diferentes para **svn merge**, y esta sección describe los más comunes que posiblemente usará alguna vez.

Fusionando una rama completa con otra

Para completar el ejemplo que hemos ido mostrando, nos moveremos hacia adelante en el tiempo. Suponga que han pasado varios días, y se han realizado muchos cambios tanto en el tronco como en su rama privada. Suponga que ya ha terminado de trabajar en su rama privada; la característica o corrección está finalmente completada, y ahora quiere fusionar todos los cambios de su rama en el tronco para que otros puedan disfrutarlos.

¿Cómo usamos **svn merge** en esta situación? Recuerde que este comando compara dos árboles, y aplica las diferencias en una copia local. Así que para recibir los cambios, necesita tener una copia local del tronco. Asumiremos que todavía tiene una copia original del tronco por alguna parte (completamente actualizada), o que recientemente ha obtenido una copia local fresca de `/calc/trunk`.

¿Pero qué dos árboles deberían ser comparados? A primera vista, la respuesta puede parecer obvia: simplemente compare el último árbol del tronco con el último árbol de su rama. ¡Pero cuidado—esta asunción es *incorrecta*, y ha confundido a muchos usuarios nuevos! Dado que **svn merge** funciona como **svn diff**, comparar el tronco y su rama *no* describirá meramente el conjunto de cambios realizados en su rama. Tal comparación muestra demasiados cambios: no solo mostraría la adición de los cambios en su rama, sino también *el borrado* de cambios en el tronco que nunca ocurrieron en su rama.

Para expresar únicamente los cambios que ocurrieron en su rama, necesita comparar el estado inicial de su rama con su estado final. Usando **svn log** en su rama, puede ver que ésta fue creada en la revisión 341. Y el estado final puede obtenerlo con la revisión `HEAD`. Esto significa que quiere comparar las revisiones 341 y `HEAD` del directorio de su rama, y aplicar esas diferencias a una copia local del tronco.

Un buen método para encontrar la revisión en la cual fue creada una rama (la "base" de la misma) es usar la opción `--stop-on-copy` con **svn log**. El sub comando log normalmente muestra todo cambio realizado en la rama, incluyendo los de la copia a partir de la cual fue creada. Así que normalmente verá también la historia del tronco. La opción `--stop-on-copy` detendrá la salida del comando en cuanto **svn log** detecte que la rama fue copiada o renombrada.

Así que en nuestro ejemplo continuo:

```
$ svn log --verbose --stop-on-copy \
          http://svn.example.com/repos/calc/branches/my-calc-branch
…
------------------------------------------------------------------------
r341 | usuario | 2002-11-03 15:27:56 -0600 (Thu, 07 Nov 2002) | 2 lines
Changed paths:
   A /calc/branches/my-calc-branch (from /calc/trunk:340)

$
```

Como esperábamos, el número final de revisión mostrado por este comando es la revisión en la cual `my-calc-branch` fue creado a raíz de una operación de copiado.

De nuevo, aquí está el procedimiento final de fusionado:

```
$ cd calc/trunk
$ svn update
At revision 405.

$ svn merge -r 341:HEAD http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile

$ svn status
M   integer.c
M   button.c
```

```
M   Makefile

# ...examine the diffs, compile, test, etc...

$ svn commit -m "Merged my-calc-branch changes r341:405 into the trunk."
Sending        integer.c
Sending        button.c
Sending        Makefile
Transmitting file data ...
Committed revision 406.
```

De nuevo, fíjese que el mensaje de cambios menciona de forma explícita el rango de cambios que fue fusionado en el tronco. Recuerde siempre hacer esto, porque es información crítica que necesitará más adelante.

Por ejemplo, suponga que decide continuar trabajando en su rama otra semana, para completar una mejora a una de las características originales o corregir un fallo. La revisión HEAD del repositorio es ahora 480, y está listo para realizar otra fusión desde su rama privada al tronco. Pero como se discutió en “Procedimientos ideales de fusionado”, no quiere fusionar los cambios que ya fueron fusionados; sólo quiere fusionar todo lo “nuevo” en su rama desde la última vez que fue fusionada. El truco es descubrir qué es lo nuevo.

El primer paso es ejecutar **svn log** en el tronco, y buscar un mensaje de cambios sobre la última vez que realizó una fusión desde su rama:

```
$ cd calc/trunk
$ svn log
…
------------------------------------------------------------------------
r406 | usuario | 2004-02-08 11:17:26 -0600 (Sun, 08 Feb 2004) | 1 line

Merged my-calc-branch changes r341:405 into the trunk.
------------------------------------------------------------------------
…
```

¡Ajá! Dado que todos los cambios que ocurrieron en la rama entre las revisiones 341 y 405 fueron fusionados con el tronco como la revisión 406, ahora sabe que sólo quiere fusionar los cambios de la rama posteriores a esto—comparando las revisiones 406 y HEAD.

```
$ cd calc/trunk
$ svn update
At revision 480.

# We notice that HEAD is currently 480, so we use it to do the merge:

$ svn merge -r 406:480 http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile

$ svn commit -m "Merged my-calc-branch changes r406:480 into the trunk."
Sending        integer.c
Sending        button.c
Sending        Makefile
Transmitting file data ...
Committed revision 481.
```

Ahora el tronco contiene la segunda oleada completa de cambios realizados en la rama. En este punto, puede o bien borrar su rama (discutiremos esto más adelante), o continuar trabajando en su rama y repetir este procedimiento en siguientes operaciones de fu-

sionado.

Deshaciendo cambios

Otro uso común de **svn merge** es deshacer un cambio que acaba de ser enviado al repositorio. Suponga que está trabajando felizmente en su copia local de `/calc/trunk`, y descubre que el cambio realizado hace tiempo en la revisión 303, que modificó `integer.c`, está completamente mal. Nunca debería haber llegado al repositorio. Puede usar **svn merge** para "deshacer" el cambio en su copia local, y entonces enviar las modificaciones locales al repositorio. Todo lo que necesita hacer es especificar la diferencia *al revés*:

```
$ svn merge -r 303:302 http://svn.example.com/repos/calc/trunk
U  integer.c

$ svn status
M  integer.c

$ svn diff
…
# verify that the change is removed
…

$ svn commit -m "Undoing change committed in r303."
Sending        integer.c
Transmitting file data .
Committed revision 350.
```

Una forma de pensar sobre las revisiones del repositorio es como un grupo de cambios (algunos sistemas de control de versiones los llaman *changesets*). Al usar el parámetro `-r`, puede pedirle a **svn merge** que aplique el changeset, o el rango completo de changesets sobre su copia local de trabajo. En nuestro caso para deshacer un cambio usamos **svn merge** para aplicar el changeset #303 a nuestra copia local *al revés*.

Tenga presente que deshacer un cambio de esta manera es como realizar cualquier otra operación **svn merge**, así que debería usar **svn status** y **svn diff** para confirmar que su trabajo está en el estado que usted desea, y entonces usar **svn commit** para enviar la versión final al repositorio. Tras el envío, este changeset particular ya no se verá reflejado en la revisión `HEAD`.

De nuevo puede estar preguntándose: vaya, eso no deshizo el cambio anterior, ¿verdad? El cambio todavía existe en la revisión 303. Si alguien obtiene una versión del proyecto `calc` entre las revisiones 303 y 349, seguirán viendo todavía el cambio malo, ¿verdad?

Si, en efecto. Cuando hablamos sobre "eliminar" un cambio, estamos hablando sobre eliminarlo de `HEAD`. El cambio original todavía existe en la historia del repositorio. En la mayoría de las situaciones, esto es suficiente. De todos modos la mayoría de las personas sólo están interesadas en seguir la versión `HEAD` de un proyecto. No obstante, hay casos especiales en los que realmente desearía destruir toda evidencia de un cambio. (Quizás alguien puso accidentalmente en el repositorio un documento confidencial.) Esto, tal y como verá, no es tan fácil porque Subversion fue diseñado deliberadamente para nunca perder información. Las revisiones son árboles inmutables que se construyen unos sobre otros. Eliminar una revisión de la historia podría causar un efecto dominó, creando caos en todas las revisiones siguientes y posiblemente invalidando todas las copias locales de trabajo. [3]

Resucitando elementos borrados

La mejor característica de los sistemas de control de versiones es que la información nunca se pierde. Incluso cuando borra un fichero o un directorio, puede haberse esfumado de la revisión `HEAD`, pero el objeto todavía existe en revisiones anteriores. Una de las preguntas más comunes realizadas por nuevos usuarios es, "¿Cómo puedo recuperar mi fichero o directorio antiguo?"

El primer paso es definir exactamente**qué** elemento está intentando resucitar. Aquí tiene una metáfora útil: puede pensar que todo

[3] Sin embargo, el proyecto Subversion tiene planes para implementar algún día un comando **svnadmin obliterate** que realizaría la tarea de borrar información permanentemente. Mientras tanto, lea "svndumpfilter" para una posible solución alternativa.

objeto en el repositorio existe en una especie de sistema de coordenadas bidimensional. Las abscisas marcan una revisión particular del árbol, y las ordenadas una ruta de fichero dentro de ese árbol. Así que cualquier versión de su fichero o directorio puede definirse por un par específico de coordenadas.

Subversion no tiene un directorio `Attic` como CVS, [4] por lo que necesita usar el comando **svn log** para descubrir la pareja de coordenadas exactas que quiere resucitar. Una buena estrategia es ejecutar **svn log --verbose** en un directorio que solía contener el elemento borrado. La opción `--verbose` muestra una lista de todos los elementos modificados en cada revisión; todo lo que necesita hacer es encontrar la revisión en la cual borró el fichero o directorio. Puede hacer esto visualmente, o usando otra herramienta para examinar el informe de cambios (vía **grep**, o quizás con una búsqueda incremental en un editor.)

```
$ cd parent-dir
$ svn log --verbose
…
------------------------------------------------------------------------
r808 | jose | 2003-12-26 14:29:40 -0600 (Fri, 26 Dec 2003) | 3 lines
Changed paths:
   D /calc/trunk/real.c
   M /calc/trunk/integer.c

Added fast fourier transform functions to integer.c.
Removed real.c because code now in double.c.
…
```

En el ejemplo, estamos asumiendo que está buscando el fichero borrado `real.c`. Leyendo los informes de cambios del directorio padre, hemos encontrado que este fichero fue borrado en la revisión 808. Por lo tanto la última versión del fichero existió en la revisión inmediatamente anterior a esa. Conclusión: quiere resucitar la ruta `/calc/trunk/real.c` de la revisión 807.

Eso fue la parte difícil—la investigación. Ahora que sabe lo que quiere recuperar, tiene dos posibles opciones.

Una de las opciones es usar el comando **svn merge** para aplicar la revisión 808 “al revés”. (Ya hemos discutido cómo deshacer cambios, lea “Deshaciendo cambios”.) Esto tendría el efecto de reañadir `real.c` como modificación local. El fichero se programaría para ser añadido, y tras enviar los cambios, el fichero existiría de nuevo en `HEAD`.

No obstante, en este ejemplo particular, ésta probablemente no es la mejor estrategia. Aplicar al revés la revisión 808 no solamente programaría la adición de `real.c`, ya que el mensaje de cambios indica que también se desharían ciertos cambios a `integer.c`, cosa que no desea. Ciertamente, podría fusionar al revés la revisión 808 y entonces ejecutar **svn revert** sobre la modificación local de `integer.c`, pero esta técnica no es muy escalable. ¿Qué pasa si hay 90 ficheros modificados en la revisión 808?

Otra estrategia más precisa es usar **svn copy** en lugar de **svn merge**. Simplemente copie la “pareja de coordenadas” exactas de revisión y ruta del repositorio a su copia de trabajo local:

```
$ svn copy --revision 807 \
           http://svn.example.com/repos/calc/trunk/real.c ./real.c

$ svn status
A  +   real.c

$ svn commit -m "Resurrected real.c from revision 807, /calc/trunk/real.c."
Adding         real.c
Transmitting file data .
Committed revision 1390.
```

El símbolo de suma en la salida del comando indica que el elemento no sólo se ha programado para ser añadido, sino para ser añadido “con historia.” Subversion recuerda de dónde fue copiado. En el futuro, ejecutar **svn log** sobre este fichero recorrerá su historia hasta la resurrección y continuará también con la que tenía antes de la revisión 807. En otras palabras, este nuevo `real.c` no es realmente nuevo; es un descendiente directo del fichero original que fue borrado.

[4] Dado que CVS no hace versionado de árboles, crea un área `Attic` dentro de cada directorio del repositorio para poder recordar los ficheros borrados.

A pesar de que nuestro ejemplo muestra cómo resucitar un fichero, tenga en cuenta que esta misma técnica proporciona los mismos resultados al resucitar directorios borrados.

Cambiando la copia local de trabajo

El comando **svn switch** transforma una copia local de trabajo existente en una rama diferente. Aunque este comando no es estrictamente necesario para trabajar con ramas, representa un buen atajo para los usuarios. En nuestro ejemplo anterior, tras crear su rama privada, obtuvo una nueva copia local del nuevo directorio del repositorio. En su lugar, puede pedirle a Subversion que cambie su copia local de `/calc/trunk` para que refleje la ubicación de la nueva rama

```
$ cd calc

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/trunk

$ svn switch http://svn.example.com/repos/calc/branches/my-calc-branch
U   integer.c
U   button.c
U   Makefile
Updated to revision 341.

$ svn info | grep URL
URL: http://svn.example.com/repos/calc/branches/my-calc-branch
```

Tras “cambiar” a la rama, su copia local no diferirá en nada a una copia local nueva obtenida del mismo directorio. Y normalmente es más eficiente usar este comando, dado que a menudo las ramas sólo difieren entre sí en pocas cosas. El servidor manda únicamente el conjunto mínimo de cambios necesarios para hacer que su copia local de trabajo refleje el directorio de la rama.

El comando **svn switch** también permite usar la opción `--revision` (`-r`), por lo que no siempre tendrá que cambiar su copia local a la “punta” de la rama.

Por supuesto, la mayoría de los proyectos son más complicados que nuestro ejemplo `calc`, y contienen múltiples subdirectorios. Los usuarios de Subversion frecuentemente siguen un algoritmo específico al usar ramas:

1. Copiar el 'tronco' entero a un nuevo directorio rama.

2. Cambiar sólo *parte* de la copia de trabajo del tronco para que refleje esa rama.

En otras palabras, si un usuario sabe que el trabajo que quiere realizar sólo ocurrirá en un subdirectorio específico, usará **svn switch** para mover sólo ese subdirectorio a la rama. (¡A veces los usuarios cambian un único fichero de su copia local a la rama!) De ese modo, sigue pudiendo recibir modificaciones normales del 'tronco' en su copia local, pero las porciones que cambió a otra rama quedarán inmunes (a no ser que alguien envíe un cambio a esa rama). Esta característica añade una nueva dimensión al concepto de “copia local mezclada”—no sólo pueden tener las copias locales una mezcla de revisiones, sino también una mezcla de rutas de repositorio.

Si su copia local contiene un número de sub árboles cambiados a otras ubicaciones del repositorio, sigue funcionando de forma normal. Cuando la actualice, recibirá los parches propios de cada sub árbol. Cuando envíe cambios, sus cambios locales seguirán siendo aplicados como un cambio atómico único al repositorio.

Tenga en cuenta que a pesar de que su copia local de trabajo puede reflejar una mezcla de lugares de repositorio, todos estos lugares deben estar dentro del *mismo* repositorio. Los repositorios de Subversion todavía no son capaces de comunicarse unos con otros; esta característica está planeada para una versión de Subversion posterior a la 1.0.[5]

Cambios y actualizaciones

¿Se ha dado cuenta de que la salida de los comandos **svn switch** y **svn update** parecen iguales? El comando de cambio es de hecho un superconjunto del comando de actualización.

Cuando ejecuta **svn update**, está preguntándole al repositorio que compare dos árboles. El repositorio lo hace, y entonces envía una descripción de las diferencias al cliente. La única diferencia entre **svn switch** y **svn update** es que el comando de actualización siempre compara dos rutas idénticas.

Es decir, si su copia local es una réplica de `/calc/trunk`, entonces **svn update** comparará automáticamente su copia local de `/calc/trunk` con `/calc/trunk` en la revisión `HEAD`. Si está cambiando su copia local a una rama, entonces **svn switch** comparará su copia local de `/calc/trunk` a algún *otro* directorio rama en la revisión `HEAD`.

En otras palabras, una actualización mueve su copia de trabajo local a través del tiempo. Un cambio mueve su copia de trabajo local a través del tiempo *y* el espacio.

Dado que **svn switch** es esencialmente una variante de **svn update**, comparte el mismo comportamiento; cualquier modificación local en su copia de trabajo es preservada cuando obtiene nuevos datos del repositorio. Esto le permite realizar toda una variedad de trucos ingeniosos.

Por ejemplo, suponga que tiene una copia local de `/calc/trunk` y realiza un número de cambios sobre ella. De repente se da cuenta de que se suponía que tenía que realizar esos cambios en una rama. ¡No hay problema! Cambie su copia local del tronco a la rama con **svn switch**, y seguirá manteniendo sus cambios locales. Ahora puede probarlos y enviar los cambios a la rama del repositorio.

Etiquetas

Otro concepto común en el control de versiones es la *etiqueta*. Una etiqueta no es más que una “instantánea” del proyecto en el tiempo. En Subversion, esta idea ya está presente en todas partes. Cada revisión del repositorio es exactamente eso—una instantánea del sistema de ficheros tras cada envío al repositorio.

No obstante, las personas prefieren dar nombres más familiares a las etiquetas, como `lanzamiento-1.0`. Y quieren hacer instantáneas de subdirectorios menores del sistema de archivos. Después de todo, no es tan fácil recordar que lanzamiento-1.0 de una parte del software es un subdirectorio particular de la revisión 4822.

Creando una etiqueta simple

De nuevo, **svn copy** viene al rescate. Si desea crear una instantánea de `/calc/trunk` exactamente tal y como se ve en la revisión `HEAD`, haga una copia:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/tags/release-1.0 \
      -m "Tagging the 1.0 release of the 'calc' project."

Committed revision 351.
```

Este ejemplo asume que ya existe un directorio `/calc/tags`. (Si no existe, vea svn mkdir). Tras finalizar la copia, el nuevo directorio `lanzamiento-1.0` será para siempre una instantánea del aspecto que tenía la revisión `HEAD` del proyecto en el momento que realizó la copia. Por supuesto puede querer ser más preciso sobre qué revisión exacta copiar, en caso de que alguien realice un cambio al proyecto mientras no está observando. Así que si sabe que la revisión 350 de `/calc/trunk` es exactamente la instantánea que desea, puede especificarla pasando `-r 350` al comando **svn copy**.

[5] Usted *puede*, no obstante, usar **svn switch** con el parámetro `--relocate` si la URL de su servidor cambia y no desea abandonar su copia de trabajo existente. Vea la sección **svn switch** en Capítulo 9, *Referencia completa de Subversion* para obtener más información y un ejemplo.

Pero espere un momento: ¿no es este procedimiento para crear etiquetas el mismo que usamos para crear una rama? Si, de hecho, así es. En Subversion no hay diferencia entre una etiqueta y una rama. Ambas son directorios ordinarios que son creados a partir de una copia. Igual que con las ramas, la única razón por la que un directorio copiado es una “etiqueta” es porque los *humanos* han decidido tratarlo de ese modo: mientras nadie envíe cambios usando ese directorio, seguirá siendo siempre una instantánea. Si la gente comienza a realizar cambios, se convierte en una rama.

Si está administrando un repositorio, hay dos formas de de gestionar etiquetas. La primera es “manos fuera”: según la política de su proyecto, decida dónde pondrá sus etiquetas, y asegúrese de que todos los usuarios saben cómo tratar los directorios que copian ahí. (En otras palabras, asegúrese de que no cambian nada en esos directorios.) La segunda forma es más paranoica: puede usar uno de los scripts de control de acceso proporcionados con Subversion para prevenir cualquier cosa que no sea la creación de nuevas copias en el área de etiquetado (Vea Capítulo 6, *Configuración del servidor*.) No obstante, el método paranoico normalmente no es necesario. Si un usuario envía un cambio accidentalmente al directorio de etiquetas, puede simplemente deshacer el cambio tal y como discutimos en la sección anterior. Después de todo, esto es control de versiones.

Creando una etiqueta compleja

A veces quiere que su “instantánea” sea más complicada que un solo directorio de una revisión concreta.

Por ejemplo, pretendamos que su proyecto es mucho más grande que nuestro ejemplo `calc`: suponga que contiene un número de subdirectorios y muchos más ficheros. Realizando su trabajo, puede decidir que necesita crear una copia de trabajo local diseñada para contener características o correcciones de fallos específicas. Puede conseguir esto modificando ficheros o directorios de forma selectiva para que reflejen una revisión particular (usando **svn update -r** liberalmente), o cambiando ficheros y directorios a ramas particulares (haciendo uso de **svn switch**). Cuando ha finalizado, su copia local es un batiburrillo de ubicaciones del repositorio de diferentes versiones. Pero tras hacer unas pruebas, sabe que es la combinación precisa de datos que necesita.

Hora de tomar una instantánea. Copiar una URL en otra no funcionará esta vez. En este caso, quiere hacer una instantánea de la configuración exacta de su copia local y almacenarla en el repositorio. Afortunadamente, **svn copy** tiene en realidad cuatro usos diferentes (sobre los cuales puede leer en el noveno capítulo), incluyendo la habilidad para copiar el árbol de una copia local al repositorio:

```
$ ls
my-working-copy/

$ svn copy my-working-copy http://svn.example.com/repos/calc/tags/mytag

Committed revision 352.
```

Ahora existe un nuevo directorio en el repositorio, `/calc/tags/mytag`, el cual es una instantánea exacta de su copia local de trabajo—revisiones mezcladas, urls, y todo eso.

Otros usuarios han encontrado usos interesantes de esta característica. A veces hay situaciones en las que acaba de enviar al repositorio un grupo de cambios locales, y querría que los viese un colaborador. En lugar de ejecutar **svn diff** y enviar el fichero de parche (el cual no captura cambios en el árbol), puede usar en su lugar **svn copy** para “subir” su copia local a un área privada del repositorio. Entonces su colaborador puede o bien obtener una copia tal cual de su copia local, o bien usar **svn merge** para recibir sus cambios exactos.

Mantenimiento de ramas

Habrá notado que Subversion es extremadamente flexible. Dado que implementa ramas y etiquetas mediante el mismo mecanismo (copias de directorio), y tanto las ramas como las etiquetas aparecen en el espacio normal del sistema de archivos, mucha gente encuentra Subversion intimidante. Es incluso *demasiado* flexible. En esta sección le ofrecemos algunas sugerencias para agrupar y gestionar sus datos a lo largo del tiempo.

Estructura del repositorio

Hay maneras estándar recomendadas para organizar un repositorio. La mayoría de las personas crean un directorio `trunk` que contendrá la "línea principal" de desarrollo, un directorio `branches` que contendrá copias de las ramas, y un directorio `tags` que contendrá copias de las etiquetas. Si un repositorio sólo almacena un proyecto, entonces la gente a menudo crea estos directorios en el directorio raíz:

```
/trunk
/branches
/tags
```

Si un repositorio contiene múltiples proyectos, los administradores normalmente indexan la estructura por proyecto (vea "Escogiendo el esquema de repositorio" para leer más sobre "raíces de proyecto"):

```
/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags
```

Por supuesto, es libre de ignorar estas estructuras comunes. Puede crear cualquier tipo de variación, lo que sea mejor para usted y su equipo. Recuerde que sea lo que sea lo que elija, no es una decisión final. Puede reorganizar su repositorio en cualquier momento. Dado que las ramas y etiquetas son directorios ordinarios, el comando **svn move** puede moverlos o renombrarlos como desee. Cambiar de una estructura a otra es una cuestión de ejecutar una serie de movimientos en el servidor; si no le gusta cómo se organizan las cosas en el repositorio, simplemente juegue un poco con los directorios.

Recuerde, no obstante, que aunque mover directorios puede ser fácil, necesita ser considerado con sus usuarios. Sus movimientos pueden desorientar a los usuarios que tengan copias locales de trabajo. Si un usuario tiene una copia local de un directorio particular del repositorio, su operación **svn move** quizás elimine la ruta de la última revisión. Cuando el usuario ejecute **svn update**, se le informará que su copia local usa una ruta que ya no existe, y el usuario se verá forzado a hacer **svn switch** a otra ubicación.

Longevidad de los datos

Otra buena característica del modelo de Subversion es que las ramas y las etiquetas pueden tener vidas infinitas, igual que cualquier otro elemento versionado. Por ejemplo, suponga que finalmente termina todo su trabajo en una rama personal del proyecto `calc`. Tras fusionar todos sus cambios de vuelta en `/calc/trunk`, ya no hace falta que el directorio de su rama privada permanezca visible:

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
             -m "Removing obsolete branch of calc project."

Committed revision 375.
```

Y desapareció su rama. Por supuesto no ha desaparecido realmente: el directorio simplemente falta en la revisión `HEAD`, y ya no distrae a nadie más. Si usa **svn checkout**, **svn switch**, o **svn list** para examinar una revisión anterior, seguirá pudiendo ver su antigua rama.

Si navegar por su directorio borrado no es suficiente, siempre puede devolverle la vida. Resucitar datos es muy fácil en Subversion. Si hay un directorio (o fichero) borrado que querría volver a poner en `HEAD`, simplemente cópielo con **svn copy -r** de la revisión antigua:

```
$ svn copy -r 374 http://svn.example.com/repos/calc/branches/my-calc-branch \
                  http://svn.example.com/repos/calc/branches/my-calc-branch
```

```
Committed revision 376.
```

En nuestro ejemplo, su rama personal tuvo un periodo de actividad relativamente corto: puede haberla creado para corregir un fallo o implementar una nueva característica. Cuando su tarea está terminada, así lo está su rama. No obstante, en el desarrollo de software también es común tener dos ramas “principales” que viven en paralelo por largos periodos de tiempo. Por ejemplo, suponga que es el momento de hacer público el proyecto estable `calc`, y sabe que va a necesitar un par de meses corregir posibles fallos del mismo. No quiere que la gente añada nuevas características al proyecto, pero tampoco quiere decirle a los desarrolladores que dejen de programar. Así que crea una rama “estable” del software que no cambiará mucho:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
         http://svn.example.com/repos/calc/branches/stable-1.0 \
         -m "Creating stable branch of calc project."

Committed revision 377.
```

Y ahora los desarrolladores son libres de continuar añadiendo características novísimas (o experimentales) a `/calc/trunk`, y usted puede declarar como política del proyecto que sólo las correcciones de fallos serán realizadas en `/calc/branches/stable-1.0`. Es decir, a medida que la gente continúa su trabajo en el tronco, un humano portará selectivamente las correcciones de fallos a la rama estable. Incluso después de haber lanzado la rama estable, probablemente continuará manteniéndola por un largo tiempo—es decir, tanto como desee continuar el soporte de esa versión para sus usuarios.

Sumario

Hemos cubierto mucho terreno en este capítulo. Hemos discutido los conceptos de las etiquetas y ramas, y demostrado como Subversion implementa estos conceptos por medio de la copia de directorios y el comando **svn copy**. Hemos mostrado cómo usar **svn merge** para copiar cambios de una rama a otra, o deshacer cambios erróneos. Hemos revisado el uso de **svn switch** para crear copias de trabajo con mezcla de ubicaciones del repositorio. Y hemos hablado sobre cómo podría gestionar la organización y tiempos de vida de las ramas en un repositorio.

Recuerde el mantra de subversion: las ramas y etiquetas son ligeras. ¡Úselas sin pudor!

Capítulo 5. Administración del Repositorio

El repositorio de Subversion es el almacén central de datos y versiones para un número indeterminado de proyectos; de esta manera se convierte en el destinatario obvio de todo el amor y atención que un administrador puede ofrecer. Aunque el repositorio es, en general, un elemento de bajo mantenimiento, es importante entender cómo configurarlo y cuidarlo correctamente para evitar problemas potenciales y para solucionar los problemas actuales de forma segura.

En este capítulo, explicaremos cómo crear y configurar un repositorio Subversion, y cómo publicarlo para acceder a él por red. También hablaremos acerca del mantenimiento del repositorio, incluyendo el uso de las herramientas **svnlook** y **svnadmin** (que están disponibles junto con Subversion). Trataremos también algunas preguntas y errores comunes, y haremos sugerencias sobre la forma de organizar los datos en el repositorio.

Si planea acceder al repositorio Subversion sólo desde el punto de vista de un usuario cuyos datos están bajo control de versiones (es decir, un cliente Subversion), puede saltarse este capítulo en conjunto. Sin embargo, si vd. es, o quiere convertirse en un administrador de repositorios Subversion, [1] debería sin lugar a dudas prestar atención a este capítulo.

Cuestiones básicas acerca de el repositorio

Antes de entrar en el tema principal de la administración del repositorio, vamos a definir con más detalle qué es un repositorio. ¿Qué pinta tiene? ¿Cómo se siente? ¿Se toma el té caliente o helado, dulce, y con limón? Como administrador, necesitará entender la composición de un repositorio desde una perspectiva lógica— entendiendo cómo se representa la información dentro del repositorio—y desde una perspectiva práctica—qué apariencia tiene y cómo actúa un repositorio con respecto a herramientas no pertenecientes a Subversion. La siguiente sección se ocupa de algunos de estos conceptos básicos a un nivel muy alto.

Entendiendo las Transacciones y Revisiones

Hablando conceptualmente, un repositorio Subversion es una secuencia de árboles de directorios. Cada árbol es una fotografía de cómo eran los ficheros y directorios versionados en tu repositorio en un momento determinado. Estas fotografías son generadas como resultado de operaciones de programas cliente, y son llamadas revisiones.

Cada revisión nace como un árbol de transacciones. Cuando se envían cambios al repositorio, el programa cliente construye una transacción de Subversion que copia los cambios locales (junto a cualquier cambio adicional que haya podido tener lugar desde el comienzo del proceso de envío de datos), y luego pide al repositorio que guarde ese árbol como la próxima fotografía en la secuencia. Si el envío de datos no da error, la transacción se convierte en una nueva revisión del árbol, y se le asigna un nuevo número de revisión. Si el envío de datos fallara por alguna razón, la transacción se destruye, y se le informa al cliente del error.

Las actualizaciones funcionan de una manera parecida. El Cliente prepara un árbol de transacción temporal que copia el estado de la copia de trabajo. El repositorio compara entonces ese árbol de transacción con el árbol de la revisión solicitada (normalmente la más reciente, o el árbol “más joven”), e informa al cliente acerca de qué cambios son necesario para convertir su copia local de trabajo en una réplica de ese árbol de revisión. Tras completarse la actualización, se borra la transacción temporal.

El uso de árboles de transacción es la única manera de hacer cambio permanentes en un repositorio de sistema de ficheros versionados. De todas maneras, es importante entender que el tiempo de vida de una transacción es completamente flexible. En el caso de actualizaciones, las transacciones con árboles temporales que se destruyen inmediatamente. En el caso de envíos al repositorio, las transacciones son transformadas en revisiones permanentes (o borradas si el envío falla). En el caso de un error, es posible que una transacción permanezca accidentalmente suelta en el repositorio (sin que afecte en realidad a nada, pero ocupando espacio).

En teoría, un día los programas de flujo de trabajo completo deberán girar hacia un control más fino del tiempo de vida de la transacción. Es factible imaginar un sistema por el que cada transacción que se convierta en revisión permanezca en bastante después de que el cliente termine de describir sus cambios al repositorio. Esto permitiría que cada nuevo envío sea revisado por alguna otra

[1] Puede sonar realmente prestigioso y , pero sólo estamos hablando de cualquier personal que esté interesada en el misterioso reino más allá de la copia de trabajo, donde los datos de todos

persona, quizás un director o , que pueda elegir entre promoverla a una revisión, o cancelarla.

Propiedades no versionadas

Las transacciones y las revisiones en el repositorio Subversion pueden tener propiedades adjuntas. Estas propiedades son relaciones genéricas clave-valor, y generalmente se usan para guardar información acerca del árbol al que están adjuntas. Los nombres y valores de estas propiedades se guardan en el sistema de ficheros del repositorio, junto con el resto de los datos de tu árbol.

Las propiedades de revisiones y transacciones son útiles para asociar información con un árbol que no está estrictamente relacionada con los ficheros y directorios de ese árbol—el tipo de información que no es gestiona por las copias de trabajo de cliente. Por ejemplo, cuando una nueva transacción de envío es creada en el repositorio, Subversion añade una propiedad a dicha transacción llamada `svn:date`— una marca de tiempo que representa el momento en que la transacción se creó. En el momento que el proceso de envío termina, el árbol también ha recibido una propiedad para guardar el nombre del usuario que es autor de la revisión (`svn:author`) y una propiedad para guardar el mensaje de informe de cambios adjunto a dicha revisión (`svn:log`).

Las propiedades de revisiones y transacciones son *propiedades no versionada*—cuando son modificadas, sus valores previos se descartan definitivamente. Así mismo, mientras los árboles de revisiones en sí son inmutables, las propiedades adjuntas de dichos árboles no lo son. Puedes añadir, borrar, y modificar propiedades de revisiones en cualquier momento más adelante. Si envías al repositorio una nueva revisión y más tarde te das cuenta de alguna información incorrecta o un error sintáctico en tu mensaje de log, puedes simplemente sustituir el valor de la propiedad `svn:log` con un nuevo y corregido mensaje de log.

Base de datos Berkeley

Los datos almacenados dentro de repositorios Subversion, realmente se encuentran en una base de datos, más concretamente, un fichero de base de datos Berkeley. Durante la fase inicial de diseño de Subversion, los desarrolladores decidieron usar una base de datos Berkeley por una serie de razones, como su licencia open-source, soporte de transacciones, ser de confianza, funcionamiento, simplicidad de su API, soporte de hilos, cursores, y más.

La base de datos Berkeley tiene un soporte real de transacciones —probablemente es su característica más poderosa. Muchos procesos que acceden a sus repositorios Subversion no tienen que preocuparse por pisotear los datos de otros. El aislamiento provisto por el sistema de transacciones es tal que por cada operación dada, el código de repositorio Subversion tiene una vista estática de la base de datos—no una base de datos que está constantemente cambiando de la mano de algunos otros procesos—y puede tomar decisiones basándose en esa vista. Si dicha decisión está en conflicto con lo que otro proceso esté haciendo, la operación completa como si nunca hubiera sucedido, y Subversion reintenta la operación contra una nueva y actualizada (y estática) vista de la base de datos.

Otra gran característica de la base de datos Berkeley son las *copias de seguridad en caliente*— la habilidad para hacer una copia de seguridad del entorno de la base de datos sin que tenga que estar “”. Hablaremos sobre cómo hacer copias de seguridad de tu repositorio en “Copias de seguridad del repositorio”, pero los beneficios de ser capaz de hacer copias completas y funcionales de tus repositorios sin debería ser obvia.

La base de datos Berkeley también es un sistema de bases de datos de mucha confianza. Subversion utiliza las utilidades de registro de las bases de datos Berkeley, lo que significa que la base datos primero escribe una descripción de cualquier modificación que vaya a hacer en ficheros de registros, para luego hace la propia modificación. Esto es para asegurar que si algo fuese mal, el sistema de base de datos pueda retroceder a un *checkpoint*—una posición en los ficheros de registro que se sabe que no están corruptas—y repetir transacciones hasta que los datos estén en un estado usable. Ver “Gestionando el espacio de almacenamiento” si quieres más información acerca de los ficheros de registro de las bases de datos de Berkeley.

Pero toda rosa tiene su espina, así que tenemos que hablar sobre algunas conocidas limitaciones de la base de datos Berkeley. Primero, los entornos de base de datos Berkeley no son portables. No puedes copiar simplemente un repositorio Subversion que fue creado en un sistema Unix a un sistema Windows y esperar que funcione. A pesar de que la mayor parte del formato de base de datos Berkeley es independiente de la arquitectura, hay otros aspectos del entorno que no lo son. Segundo, Subversion usa la base de datos Berkeley de una manera que no puede funcionar en sistemas Windows 95/98—si necesita almacenar un repositorio en una máquina Windows, utilice Windows 2000 o Windows XP. Finalmente, no deberías mantener un repositorio Subversion en una unidad compartida por red. Mientras las bases de datos Berkeley prometen un comportamiento correcto en unidades compartidas por red que cumplan un grupo particular de especificaciones, casi ningún sistema de compartición conocido cumple con todas esas especificaciones.

Creación y Configuración de Repositorios

Crear un repositorio Subversion es una tarea increíblemente simple. La utilidad **svnadmin**, provista con Subversion, tiene un subcomando justo para esto. Para crear un nuevo repositorio, ejecuta:

```
$ svnadmin create /path/to/repos
```

Crea un nuevo repositorio en el directorio `/path/to/repos`. Dicho nuevo repositorio comienza su vida en la revisión 0, que se define como nada excepto el directorio raíz (`/`) del sistema de ficheros. Inicialmente, la revisión 0 tiene también una única propiedad de revisión, `svn:date`, que tiene la hora a la que el el repositorio fue creado.

No crees tu repositorio en una unidad de red compartida —no *puede* existir un un sistema de ficheros remoto como NFS, AFS, o Windows SMB. La base de datos Berkeley necesita que el sistema de ficheros subyacente implemente estrictamente la semántica de bloqueo POSIX, y más importante, la habilidad para mapear ficheros directamente Casi ningún sistema de ficheros de red tiene estas características. Si intentas usar una base de datos Berkeley en una unidad compartida de red, los resultados son impredecibles —puede que veas errores misteriosos, o pueden pasar meses hasta que descubras que la base de datos de tu repositorio está sutilmente corrupta.

Si necesitas que varios ordenadores accedan al repositorio, deberías instalar un proceso servidor (como Apache o **svnserve**), almacenar el repositorio en un sistema de ficheros local al que el servidor pueda acceder, y hacer que el repositorio esté disponible por la red. Capítulo 6, *Configuración del servidor* se ocupa de este proceso en detalle.

Te habrás dado cuenta de que el argumento de ruta de **svnadmin** fue sólo una ruta normal del sistema de ficheros y no una URL como la que el programa cliente **svn** usa cuando se refiere a los repositorios. Tanto **svnadmin** como **svnlook** son considerados como utilidades del lado del servidor— se usan en la máquina donde reside el repositorio para examinar o modificar aspectos del mismo, y son de hecho, tareas imposibles de realizar por red. Un error común hecho por recién llegados a Subversion es tratar de pasar URLs (incluso las `file:` "locales") a ambos programas.

Así, después de que hayas ejecutado el comando **svnadmin create**, tienes un nuevo y brillante repositorio Subversion en su propio directorio. Echemos una ojeada a qué es lo que realmente se crea dentro de ese subdirectorio.

```
$ ls repos
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

Con la excepción de los ficheros `README.txt` y `format`, el directorio del repositorio es un grupo de subdirectorios. Al igual que en otras áreas del diseño de Subversion, se le tiene mucho respeto a la modularidad, y se prefiere una organización jerárquica antes que un caos que estorbe. He aquí una breve descripción de todos los objetos que puedes ver en tu nuevo directorio de repositorio:

conf

Un directorio que contiene los ficheros de configuración del repositorio.

dav

Un directorio para Apache y mod_dav_svn y su economía privada de datos.

db

El entorno principal de la base de datos Berkeley, lleno de tablas que el almacenamiento de datos para el sistema de ficheros de Subversion (donde todos tus datos versionados residen.

format

Un fichero cuyo contenido es un simple valor entero que nos dice el número de versión del repositorio

hooks

Un directorio lleno de plantillas de ganchos (y los mismos ganchos), una vez que hayas instalados algunos.

locks

Un directorio para el bloqueo de datos de repositorio de Subversion, usado para los accesos al repositorio.

README.txt

Un fichero que simplemente informa a sus lectores que están mirando un repositorio Subversion.

En general, no deberías con tu repositorio "a mano". La herramienta **svnadmin** debería ser suficiente para cualquier cambio necesarios en tu repositorio, o puedes echar una ojeada a herramientas de terceros (como la suitede herramientas de la base de datos Berkeley) para subsecciones relevantes del repositorio. Sin embargo, hay algunas excepciones, y las veremos aquí.

Scripts de enganche

Un *gancho* es un programa activado por algún evento del repositorio, como la creación de una nueva revisión o la modificación de una propiedad no versionada. A cada gancho se le da suficiente información para que sepa de qué evento se trata, cuál es su objetivo, y el nombre de usuario de la persona que disparó el evento. Dependiendo de la salida del gancho o de estado de su salida, el programa de enganche puede continuar la acción, pararla, o suspenderla de alguna manera.

El subdirectorio `hooks` se rellena, por defecto, con plantillas para varios ganchos de repositorio.

```
$ ls repos/hooks/
post-commit.tmpl          pre-revprop-change.tmpl
post-revprop-change.tmpl  start-commit.tmpl
pre-commit.tmpl
```

Hay una plantilla por cada gancho que implementa el repositorio Subversion, y examinando los contenidos de dichas plantillas de scripts, puede ver qué disparador ejecuta cada script y qué datos se le pasan. También se encuentran en muchas de estas plantillas ejemplos de cómo debería usar dicho script, conjuntamente con otros programas provistos por Subversion, para realizar tareas comunes y útiles. Para instalar realmente un gancho funcional, sólo necesita colocar algún ejecutable o script en el directorio `repos/hooks` que pueda ser ejecutado con el nombre del gancho (como **start-commit** o **post-commit**).

En plataformas Unix, esto significa proveer un script o programa (podría ser un shell script, un programa Python, un binario C compilado, o cualquier otra cosa) llamado exactamente igual que el nombre del gancho. Por supuesto, los ficheros de plantillas están presentes para algo más que sólo propósitos informativos—la manera más fácil de instalar un gancho en plataformas Unix es simplemente copiar el fichero de plantilla apropiado en un nuevo fichero sin la extensión `.tmpl`, personalizando los contenidos del gancho, y asegurándose de que el script sea ejecutable. Windows, sin embargo, usa extensiones de fichero para determinar si un programa es ejecutable o no, así que debería dar poner un programa cuyo nombre sea el nombre del gancho, y cuya extensión sea una de las extensiones especiales reconocidas por Windows como ficheros ejecutables, como `.exe` o `.com` para programas, y `.bat` para ficheros de scripts.

Actualmente hay cinco ganchos implementados por el repositorio Subversion:

`start-commit`

Se ejecuta antes de la transacción de envío haya sido creada. Se usa normalmente para decidir si el usuario tiene los privilegios suficientes. El repositorio pasa dos argumentos a este programa: la ruta al repositorio, y el nombre de usuario que intenta realizar el envío. Si el programa devuelve algún valor distinto de cero, se paraliza el envío antes de haber creado la transacción.

`pre-commit`

Se ejecuta cuando se completa la transacción, pero antes de ser enviados los datos al repositorio. Este gancho se usa normalmente como protección contra envíos que no se permiten debido a contenido o ubicación (por ejemplo, su sitio puede requerir que todos los cambios sobre una rama determinada incluyan un número de ticket del seguimiento de fallos, o que el mensaje de informe de cambios entrante no esté vacío). El repositorio pasa dos argumentos a este programa: la ruta al repositorio, y el nombre de la transacción que va a sufrir el cambio. Si el programa devuelve una salida que no sea cero, el envío se aborta y la transacción se borra.

La distribución Subversion incluye algunos scripts de control de acceso (ubicados en el directorio `tools/hook-scripts` del árbol de fuentes de Subversion) a los que se puede llamar desde **pre-commit** para implementar un control de de acceso más en detalle. En estos momentos, esta es la única manera por la cual los administradores pueden controlar el acceso más allá de lo que el fichero `httpd.conf` de Apache ofrece. En una versión futura de Subversion, planeamos implementar listas de control de acceso (ACLs) directamente en el sistema de ficheros.

`post-commit`

Esto se ejecuta después de que la transacción se haya confirmado, y una nueva revisión haya sido creada. La mayoría de la gente usa este gancho para enviar correos descriptivos acerca de la confirmación o para hacer una copia de seguridad del repositorio. El repositorio pasa dos argumentos a este programa: la ruta al repositorio, y el número de la nueva revisión creada. El código de salida del programa es ignorado.

La distribución de Subversion incluye un script **commit-email.pl** (ubicado en el directorio `tools/hook-scripts/` del árbol de fuentes de Subversion) que puede ser usado para enviar correos electrónicos con (añadiendo o no un fichero de log) una descripción de la confirmación hecha. Este correo contiene una lista de las rutas que fueron cambiadas, el mensaje de log adjunto a la confirmación, el autor y fecha de la confirmación, así como un informe en formato de GNU diff de los cambios hechos en los ficheros versionados como parte de la confirmación.

Otra herramienta útil de Subversion es el script **hot-backup.py** (ubicado en el directorio `tools/backup/` del árbol de fuentes de Subversion). Este script realiza copias de seguridad en caliente de su repositorio Subversion (una capacidad soportada por el motor de bases de datos Berkeley), y puede ser usada para hacer una foto por cada confirmación de su repositorio para archivar, o con el objetivo de recuperación de emergencia.

`pre-revprop-change`

Al no ser versionadas las propiedades de revisión de Subversion, hacer modificaciones a una de ellas (como por ejemplo, el mensaje de informe de cambios `svn:log`) sobreescribirá el valor previo de esa propiedad para siempre. Como hay datos aquí que potencialmente se pueden perder, Subversion provee este gancho (y su contrapartida, `post-revprop-change`) de tal manera que los administradores de repositorios puedan mantener con métodos externos si así lo desean, registros de los cambios de dichas propiedades. Como precaución contra la pérdida de datos de propiedades no versionadas, no se permite a los clientes Subversion modificarlos del todo remotamente a no ser que este gancho se implemente para su repositorio.

Este gancho se ejecuta justo antes de que una modificación de este tipo se haga en el repositorio. El repositorio pasa cuatro argumentos al gancho: la ruta al repositorio, la revisión en la cual la propiedad que se va a modificar existe, el nombre autenticado de usuario de la persona que va a hacer el cambio, y el nombre mismo de la propiedad.

`post-revprop-change`

Como se ha mencionado antes, este gancho es la contrapartida del gancho `pre-revprop-change`. De hecho, por paranoia, este script no se ejecutará a no ser que el gancho `pre-revprop-change` exista. Cuando ambos están presentes, el gancho `post-revprop-change` se ejecuta justo después de que una propiedad de revisión haya sido modificad, y se usa típicamente para enviar un correo electrónico con el nuevo valor de la propiedad cambiada. El repositorio pasa cuatro argumentos al gancho: la ruta al repositorio, la revisión en la cual la propiedad existe, el nombre autenticado de usuario de la persona que va a hacer el cambio y el nombre mismo de la propiedad.

La distribución de Subversion incluye el script **propchange-email.pl** script (ubicado en el directorio `tools/hook-scripts/` del árbol de fuentes de Subversion) que puede ser usado para enviar un correo electrónico con (y/o añadido a un fichero de log) los detalles de un cambio en una propiedad de revisión. Este correo electrónico contiene la revisión y nombre de la propiedad modificada, el usuario que hizo el cambio, y el nuevo valor.

Subversión tratará de ejecutar los ganchos como el mismo usuario que posee el proceso que está accediendo al repositorio Subversion. En la mayoría de los casos, el repositorio se accede a través del servidor HTTP Apache y mod_dav_svn, así que este usuario es el mismo que el usuario con el que se ejecuta Apache. Los mismos ganchos necesitarán ser configurados con permisos a nivel de sistema operativo que les permitan ser ejecutados por dicho usuario. Asimismo, esto significa que cualquier fichero o programas (incluyendo el repositorio mismo) al que acceda directa o indirectamente el gancho, será a través del mismo usuario. En otras palabras, esté alerta a problemas potenciales relacionados con permisos que impidan al gancho realizar las tareas para las cuales lo haya escrito.

Configuración de la base de datos Berkeley

Un entorno de base de datos Berkeley es una encapsulación de una o más bases de datos, ficheros de registro, ficheros regionales y ficheros de configuración. El entorno de base de datos Berkeley tiene su propio conjunto de valores de configuración por defecto para cosas como el número de bloqueos que se permite eliminar de una sola vez, o el tamaño máximo de los ficheros de registro , etc. El código del sistema de ficheros de Subversion elige además valores por defecto para algunas de las opciones de configuración de la base de datos Berkeley. De todas maneras, algunas veces su repositorio particular, con su colección única de datos y patrones de acceso, podría necesitar un grupo diferente de valores de configuración.

La gente de Sleepycat (los programadores de la base de datos Berkeley) entienden que bases de datos diferentes, tienen requerimientos específicos, de tal manera, que nos proveen de un mecanismo para sobreescribir en tiempo de ejecución, muchos de los valores de configuración del entorno Berkeley. Berkeley comprueba la presencia de un fichero denominado `DB_CONFIG` en cada directorio del entorno, y las opciones que encuentra en dicho fichero para usarlas en ese entorno Berkeley particular.

El fichero de configuración de Berkeley para su repositorio se encuentra en directorio del entorno `db`, en `repos/db/DB_CONFIG`. Subversion crea por sí mismo el fichero al mismo tiempo que el resto del repositorio. El fichero inicialmente contiene algunas opciones por defecto, así como enlaces a la documentación en línea de la base de datos Berkeley de tal manera que pueda saber qué significan dichas opciones. Por supuesto, es libre de añadir cualquiera de las opciones soportadas por la base de datos Berkeley a su fichero `DB_CONFIG`. Tan sólo tenga cuidado porque mientras Subversion no trata de leer o interpretar los contenidos del fichero, y no hace uso de sus opciones de configuración, tendrá que evitar cualquier cambio en la configuración que pueda causar que la base de datos Berkeley se comporte de una manera inesperada para el resto del código de Subversion. Por otra parte, los cambios hechos en `DB_CONFIG` no tendrán efecto hasta que vuelva a leer en entorno de la base de datos (usando **svnadmin recover**).

Mantenimiento del Repositorio

El mantenimiento de un repositorio Subversion puede ser una tarea que asuste, principalmente debido a las complejidades inherentes a sistemas que tienen un motor de base de datos. Para hacer bien este trabajo, basta con conocer las herramientas—qué son y cuándo y cómo usarlas. Esta sección le presentará las herramientas de administración que provee Subversion, y cómo utilizarlas para realizar tareas como migraciones de repositorios, actualizaciones, copias de seguridad y limpiezas.

Una caja de herramientas del Administrador

Subversion provee un conjunto de utilidades para crear inspeccionar, modificar y reparar sus repositorio. Estudiemos más de cerca cada una de estas herramientas. Más tarde, examinaremos brevemente otras incluidas en la distribución de la base de datos Berkeley que añaden funcionalidades específicas al motor de base de datos de las que no disponen las herramientas propias de Subversion.

svnlook

svnlook es una herramienta provista por Subversion para examinar las revisiones y transacciones realizadas en un repositorio. Este programa no intente en ningún momento cambiar el repositorio —es una utilidad de “sólo lectura”. **svnlook** es utilizada normalmente por los ganchos del repositorio para informar acerca de los cambios que se van a realizar (en el caso del gancho **pre-commit**) o que se acaban de hacer (en el caso del gancho **post-commit**) en el repositorio. Un administrador del repositorio puede usar esta herramienta para diagnosis.

svnlook tiene una sintaxis muy simple:

```
$ svnlook help
uso general. svnlook SUBCOMANDO RUTA_REPOS [PARAMS y OPCIONES ...]
Nota: todo subcomando que tome los parámetros '--revision' y '--transaction'
      actuará, si se lo invoca sin una de estas opciones, sobre la versión
      más reciente del repositorio.
Escriba "svn help <subcomando>" para ayuda en un subcomando específico.
…
```

Casi cada uno de los subcomandos de **svnlook** puede trabajar sobre una revisión o una transacción, imprimiendo información acerca del propio árbol, o de sus diferencias con respecto a la revisión anterior del repositorio. Usará la opciones `--revision` y `--transaction` para especificar qué revisión o transacción examinar respectivamente. Nótese que mientras los números de revisión se ven como números naturales, los nombres de transacción son cadenas alfanuméricas. Recuerde que el sistema de ficheros sólo permite navegar entre transacciones no confirmadas (transacción que no han tenido como resultado una nueva revisión). La mayoría de los repositorios no tendrán ese tipo de transacciones, porque las transacciones normalmente son confirmadas (lo que evita que puedan ser examinadas) o paradas y borradas.

En ausencia de las opciones `--revision` y `--transaction`, **svnlook** examinará la última revisión (o “HEAD”) del repositorio. Así que las siguientes órdenes hacen exactamente lo mismo cuando 19 es la última revisión del repositorio ubicado en `/path/to/repos`:

```
$ svnlook info /path/to/repos
$ svnlook info /path/to/repos --revision 19
```

La única excepción a estas reglas es el subcomando **svnlook youngest**, que no recibe opciones, y que simplemente escribe el número de revisión de `HEAD`.

```
$ svnlook youngest /path/to/repos
19
```

La salida de **svnlook** está diseñada para que sea, a la vez, legible por humanos y por máquinas. Cojamos como ejemplo la salida del subcomando `info`:

```
$ svnlook info /path/to/repos
sally
2002-11-04 09:29:13 -0600 (Mon, 04 Nov 2002)
27
Added the usual
Greek tree.
```

La salida del subcomando `info` está definida como:

1. El autor, seguido por un salto de línea.
2. La fecha, seguida por un salto de línea.
3. El número de caracteres en el mensaje de registro, seguido por un salto de línea.
4. El mensaje de registro, seguido por un salto de línea.

Esta salida es legible para humanos, elementos como la fecha, se presentan usando una representación de texto en lugar de algo más oscuro (como el número de nanosegundos desde que). Pero esta salida es también legible por la máquina—debido a que el registro del mensaje puede contener muchas líneas y no tener un límite de tamaño, **svnlook** informa del tamaño del mensaje antes que el mismo mensaje. Esto permite a los y otros alrededor de este comando, tomar decisiones inteligentes acerca del mensaje de registro, como cuánta memoria reservarle, o al menos, cuántos bytes saltarse en el evento para que esta salida no sea el último bit de datos en el flujo.

Otro uso común de **svnlook** es ver realmente los contenidos de un árbol de revisión o de transacción. El comando **svnlook tree** presenta los directorios y fichero del árbol solicitado. Si añade la opción `--show-ids`, también enseñará los identificadores de nodos del sistema de ficheros de la revisión para cada una de las rutas (que, en general, es más útil para los desarrolladores que para los usuarios).

```
$ svnlook tree /path/to/repos --show-ids
/ <0.0.1>
 A/ <2.0.1>
  B/ <4.0.1>
   lambda <5.0.1>
   E/ <6.0.1>
    alpha <7.0.1>
    beta <8.0.1>
   F/ <9.0.1>
  mu <3.0.1>
  C/ <a.0.1>
  D/ <b.0.1>
   gamma <c.0.1>
   G/ <d.0.1>
    pi <e.0.1>
    rho <f.0.1>
    tau <g.0.1>
   H/ <h.0.1>
    chi <i.0.1>
    omega <k.0.1>
    psi <j.0.1>
 iota <1.0.1>
```

Una vez que haya visto el esquema de los ficheros y directorios de su árbol, podrá usar comandos como **svnlook cat**, **svnlook propget**, y **svnlook proplist** para profundizar en los detalles dichos elementos.

svnlook puede realizar otros tipos de consultas, presentando los subgrupos de información que mencionamos antes, informando acerca de qué rutas fueron modificadas en una revisión o transacción especificada, mostrando las modificaciones de textos o propiedades realizadas, etc ... A continuación hay una breve descripción de la actual lista de subcomandos aceptados por **svnlook**, y la salida de los mismos:

`author`

Dice el autor del árbol.

`cat`

Vuelca los contenidos de un fichero en el árbol.

`changed`

Da una lista de todos los ficheros y directorios que han sido modificados en el árbol.

`date`

Dice la fecha del árbol.

`diff`

Vuelca de ficheros modificados.

`dirs-changed`

Lista los directorios del árbol que han sido modificados, o aquellos en los que alguno de sus ficheros han sido modificados.

`history`

Presenta puntos interesantes en la historia de una ruta versionada (lugares donde ha habido modificaciones o copias).

`info`

Vuelca el autor, la fecha, el número de caracteres del mensaje de registro, y el mensaje de registro.

`log`

Dice el mensaje de registro del árbol.

`propget`

Dice el valor de una propiedad de una ruta en el árbol.

`proplist`

Vuelca los nombres y valores de las propiedades de rutas en el árbol.

`tree`

Vuelca el listado del árbol, revelando opcionalmente el identificador de la revisión de nodo de sistema de ficheros asociado a cada ruta.

`uuid`

Dice el identificador único de usuario del árbol.

`youngest`

Dice el último número de revisión.

svnadmin

El programa **svnadmin** es el mejor amigo del administrador del repositorio. Además de darle la posibilidad de crear repositorios Subversion, le permite realizar varias operaciones de mantenimiento en ellos. La sintaxis de **svnadmin** es parecida a la de **svnlook**:

```
$ svnadmin help
uso general: svnadmin SUBCOMANDO RUTA_REPOS  [PARAMS y OPCIONES ...]
Escriba "svnadmin help <subcomando>" para ayuda sobre un subcomando.

Subcomandos disponibles:
   create
   deltify
   dump
   help (?, h)
…
```

Ya hemos mencionado antes el subcomando `create` de **svnadmin** (ver "Creación y Configuración de Repositorios"). La mayoría de los demás subcomandos los veremos más adelante en en este capítulo. De momento, veamos por encima lo que cada uno de ellos nos ofrece.

`create`

Crear un nuevo repositorio Subversion.

`deltify`

Recorre un rango de revisiones específico, realizando la deltificación de predecesores en las rutas modificadas de esas revisiones. Si no se especifica ninguna revisión, este comando simplemente deltificará la revisión `HEAD`.

`dump`

Vuelca los contenidos del repositorio, agrupados por por un conjunto dado de revisiones, usando un formato portable de volcado.

`hotcopy`

Hace una copia en caliente de un repositorio. Puede ejecutar esta comando en cualquier momento, y hacer una copia segura del repositorio sin que se vea afectada por el hecho de que otros procesos estén accediendo al repositorio.

`list-dblogs`

Lista las rutas a los ficheros de registro de la base de datos Berkeley asociados al repositorio. Esta lista incluye todos los ficheros de registros—aquellos que están todavía en uso por Subversion, así como los que no lo están.

`list-unused-dblogs`

Lista las rutas de los ficheros de registro de la base de datos Berkeley asociados al repositorio, pero que han dejado de usarse. Puede borrar de forma segura estos ficheros del entorno del repositorio y archivarlos para el caso de tener que hacer una recuperación de algún evento catastrófico del repositorio.

`load`

Carga un grupo de revisiones en un repositorio desde un flujo de datos que utilice el mismo formato portable de información que el generado por el subcomando `dump`.

`lstxns`

Lista los nombres de las transacciones Subversion no confirmadas que existen actualmente en el repositorio.

`recover`

Realiza tareas de recuperación en un repositorio que lo necesite, generalmente tras un error fatal que haya impedido a un proceso cerrar limpiamente su comunicación con el repositorio.

`rmtxns`

Borra limpiamente del repositorio transacciones Subversion (convenientemente nutrido por la salida del subcomando `lstxns`).

`setlog`

Sustituye el valor actual de la propiedad `svn:log` (mensaje de confirmación) de una transacción en particular en el repositorio con un nuevo valor.

`verify`

Verificar los contenidos del repositorio. Esto incluye, entre otras cosas, comparaciones de firmas de los datos versionados almacenados en el repositorio.

svndumpfilter

Debido a que Subversion almacena toda la información en un sistema de bases de datos opaco, intentar hacer cambios a mano no es nada recomendable, además de ser bastante complicado. Además, una vez que la información ha sido guardada en su repositorio, Subversion, en general, no tiene ninguna manera cómoda de borrarla. [2] Inevitablemente, habrá veces que querrá manipular el histórico de su repositorio. Podría necesitar eliminar todas la versiones de un fichero que se añadió accidentalmente (y que no debería estar ahí por alguna razón). O quizás tiene varios proyectos compartiendo el mismo repositorio, y ha decidido separarlos cada uno en su propio repositorio. Para realizar este tipo de tareas, los administradores necesitan una representación de los datos más manejable—el formato de volcado.

El formato de volcado del repositorio Subversion es una representación legible por humanos de los cambios hechos a tus datos versionados a lo largo del tiempo. Utiliza el comando **svnadmin dump** para generar el volcado, y **svnadmin load** para poblar un nuevo repositorio con ellos (ver “Migrando un repositorio”). Lo mejor del hecho de que el formato sea legible es que, si no te asusta, puedes revisarlo y modificarlo manualmente. Por supuesto, lo pero que es si tienes dos años de trabajo en un repositorio encapsulado en lo que se supone que será un fichero de volcado muy grande, llevará mucho, mucho tiempo el revisarlo y modificarlo.

A pesar de que no será la herramienta más usada a disposición del administrador, **svndumpfilter** tiene un producto muy peculiar con una funcionalidad muy útil—la posibilidad de modificar rápida y fácilmente esos datos de volcado actuando como un filtro basado en las rutas. Simplemente, diga una lista de rutas que quieres mantener, o una lista de rutas que no quieres mantener, luego, dirige tu volcado a través de este filtro. El resultado será un volcado modificado que contendrá sólo las rutas versionadas que (explícita o implícitamente) pediste.

La sintaxis de **svndumpfilter** es así:

```
$ svndumpfilter help
uso general: svndumpfilter SUBCOMANDO [ARGS & OPCIONES ...]
Escriba "svndumpfilter help <subcommand>" para obtener información sobre un subcomando
específico.

Subcomandos disponibles:
   exclude
   include
   help (?, h)
```

Hay sólo dos subcomandos interesantes. Te permiten elegir entre una inclusión explícita o implícita de rutas en el flujo de información:

`exclude`

Excluye del flujo de datos del volcado a un grupo de rutas.

`include`

Permite sólo al grupo de rutas solicitado, pasar al flujo de datos del volcado.

Veamos un ejemplo realista acerca de qué uso le podría dar a este programa. Más tarde hablaremos (ver “Escogiendo el esquema de repositorio”) del proceso de decisión del modelo de datos para su repositorio—un repositorio por proyecto o combinándolos, organizar todo el material dentro de su repositorio. A veces, después de que las nuevas revisiones comienzan a volar, te vuelves a pensar el modelo, y querrías hacer algunos cambios. Un cambio común es la decisión de mover proyectos que están compartiendo un sólo repositorio en un repositorio separado para cada proyecto.

[2] Esto, de todas maneras, es una *característica*, y no un error.

Nuestro repositorio imaginario contiene tres proyectos: `calc`, `calendar`, y `spreadsheet`. Han estado en una modelo como este:

```
/
   calc/
      trunk/
      branches/
      tags/
   calendar/
      trunk/
      branches/
      tags/
   spreadsheet/
      trunk/
      branches/
      tags/
```

Para meter cada uno de estos tres proyectos en su repositorio respectivo, debemos primero hacer un fichero de volcado del repositorio completo:

```
$ svnadmin dump /path/to/repos > repos-dumpfile
* Dumped revision 0.
* Dumped revision 1.
* Dumped revision 2.
* Dumped revision 3.
…
$
```

Luego, pasar ese fichero de volcado a través del filtro, incluyendo cada vez, sólo uno de los directorios superiores, dando como resultado tres nuevos ficheros de volcado:

```
$ svndumpfilter include calc < repos-dumpfile > calc-dumpfile
…
$ svndumpfilter include calendar < repos-dumpfile > cal-dumpfile
…
$ svndumpfilter include spreadsheet < repos-dumpfile > ss-dumpfile
…
$
```

En este momento, debe tomar una decisión. Cada uno de sus ficheros de volcado creará un repositorio válido, pero mantendrá las rutas exactamente como estaban en el repositorio original. Esto significa que aunque tenga un repositorio exclusivamente para su proyecto `calc`, dicho repositorio tendrá todavía un directorio superior llamado `calc`. Si quiere que los directorios `trunk`, `tags`, y `branches` estén en la raíz del repositorio, necesitará editar los ficheros de volcado, cambiando las cabeceras `Node-path` y `Copyfrom-path` para conseguir que esa primera ruta `calc/` no aparezca; además, deberá borrar la sección del volcado que crea el directorio `calc`. Será algo como:

```
Node-path: calc
Node-action: add
Node-kind: dir
Content-length: 0
```

Todo lo que queda por hacer ahora es crear sus tres nuevos repositorios, y cargar cada fichero de volcado en el repositorio correcto:

```
$ svnadmin create calc; svnadmin load calc < calc-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : button.c ... done.
…
$ svnadmin create calendar; svnadmin load calendar < cal-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : cal.c ... done.
…
$ svnadmin create spreadsheet; svnadmin load spreadsheet < ss-dumpfile
<<< Started new transaction, based on original revision 1
     * adding path : Makefile ... done.
     * adding path : ss.c ... done.
…
$
```

Ambos subcomandos de **svndumpfilter** aceptan opciones para decidir cómo tratar las revisiones “vacías”. Si una revisión dada sólo contenía cambios a rutas que fueron excluidas al filtrarlas, podría ser considerada como no interesante o incluso ignorada. Así que, para darle al usuario el control sobre qué hacer con esas revisiones, **svndumpfilter** tiene las siguientes opciones:

`--drop-empty-revs`

No generar ninguna revisión vacía— tan sólo ignorarlas.

`--renumber-revs`

Si las revisiones vacías son descartadas (usando la opción `--drop-empty-revs`), cambiar los números de revisión de las restantes de tal manera que no haya huecos en la secuencia numérica.

`--preserve-revprops`

Si las revisiones vacías no son descartadas, preservar las propiedades de revisión (mensaje de registro, autor, fecha, propiedades personalizadas, etc ...) de dichas revisiones vacías. Por otra parte, las revisiones vacías sólo contendrán la marca de tiempo original, y un mensaje de registro generado indicando que esta revisión fue vaciada por **svndumpfilter**.

Mientras **svndumpfilter** puede ser muy útil, y un gran ahorrador de tiempo, desafortunadamente hay un par de problemas. Primero, esta utilidad es . Preste atención a si las rutas en su fichero de volcado se especifican con o sin barras al principio. Necesitará mirar las cabeceras `Node-path` y `Copyfrom-path`.

```
…
Node-path: spreadsheet/Makefile
…
```

Si las rutas tienen barras al principio, deberá incluir barras al principio en las rutas que quiera pasar a **svndumpfilter include** y **svndumpfilter exclude** (si no las tienen, no debería incluirlas). Además, si su fichero de volcado tiene un uso inconsistente de estas barras por alguna razón, [3] probablemente debería normalizar esas rutas de tal manera que todas ellas tengan, o no tengan, barras al principio.

También las rutas copiadas pueden darle algunos problemas. Subversion soporta operaciones de copia en el repositorio, donde una nueva ruta se crea copiando otra ruta ya existente. Es posible que en algún momento de la vida de su repositorio, vd. haya copiado un fichero o un directorio desde alguna ubicación que **svndumpfilter** esté excluyendo a alguna otra que se esté incluyendo. Para conseguir que los datos volcados sean autosuficientes, **svndumpfilter** todavía necesita enseñar la adición de la nueva ru-

[3] Mientras **svnadmin dump** tiene una política consistente de barras—no incluirlas— otros programas que generan datos de volcado puede no ser tan consistentes.

ta—incluyendo los contenidos de los ficheros creados por la copia—y no representar dicha adición como una copia de una fuente que no existirá en el flujo filtrado de los datos de volcado. Pero debido a que el formato de volcado de repositorios Subversion sólo presenta lo modificado en cada revisión, los contenidos del origen de la copia podrían no estar disponibles para su lectura. Si sospechase que tiene alguna copia de este tipo en su repositorio, deberá repensar su grupo de rutas incluyentes y/o excluyentes.

svnshell.py

El árbol de código fuente de Subversion también trae una interfaz al repositorio tipo línea de comando. El script en Python **svnshell.py** (ubicado en `tools/examples/` en el árbol de fuentes) usa las interfaces de lenguaje de Subversion (así que debe tenerlas correctamente compiladas e instaladas para que este script funcione) para conectar al repositorio y las librerías de sistema de ficheros.

Una vez iniciado, el programa se comporta de manera similar a una línea de comando, permitiéndole navegar a través de los directorios de su repositorio. Inicialmente se le "posiciona" en el directorio raíz de la revisión `HEAD` del repositorio, y se le muestra la línea de comando. Puede usar el comando `help` en cualquier momento para leer un listado de los comandos disponibles y su propósito.

```
$ svnshell.py /path/to/repos
<rev: 2 />$  help
Available commands:
  cat FILE     : dump the contents of FILE
  cd DIR       : change the current working directory to DIR
  exit         : exit the shell
  ls [PATH]    : list the contents of the current directory
  lstxns       : list the transactions available for browsing
  setrev REV   : set the current revision to browse
  settxn TXN   : set the current transaction to browse
  youngest     : list the youngest browsable revision number
<rev: 2 />$
```

Puede navegar por la estructura de directorios de su repositorio del mimo modo que lo haría con la línea de comando de Unix o Windows—usando el comando `cd`. En todo momento la línea de comando le mostrará qué revisión (prefijada por `rev:`) o transacción (prefijada por `txn:`) está examinando, y la ruta dentro de esa revisión o transacción. Puede cambiar la revisión o transacción actual con los comandos `setrev` y `settxn` respectivamente. Al igual que en una línea de comando de Unix, puede usar el comando `ls` para mostrar el contenido del directorio actual, y puede usar el comando `cat` para mostrar el contenido de un fichero.

Ejemplo 5.1. Usando svnshell para navegar por el repositorio

```
<rev: 2 />$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     2.0.1>          Nov 15 11:50 A/
     2    harry <     1.0.2>       56 Nov 19 08:19 iota
<rev: 2 />$ cd A
<rev: 2 /A>$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     4.0.1>          Nov 15 11:50 B/
     1    sally <     a.0.1>          Nov 15 11:50 C/
     1    sally <     b.0.1>          Nov 15 11:50 D/
     1    sally <     3.0.1>       23 Nov 15 11:50 mu
<rev: 2 /A>$ cd D/G
<rev: 2 /A/D/G>$ ls
   REV   AUTHOR  NODE-REV-ID     SIZE         DATE NAME
--------------------------------------------------------------------------------
     1    sally <     e.0.1>       23 Nov 15 11:50 pi
     1    sally <     f.0.1>       24 Nov 15 11:50 rho
```

```
     1    sally <     g.0.1>       24 Nov 15 11:50 tau
<rev: 2 /A>$ cd ../..
<rev: 2 />$ cat iota
This is the file 'iota'.
Added this text in revision 2.

<rev: 2 />$ setrev 1; cat iota
This is the file 'iota'.

<rev: 1 />$ exit
$
```

Como puede ver en el ejemplo anterior, puede indicar múltiples comandos en una única línea separados por puntos y comas. Además, la línea de comando entiende los conceptos de rutas relativas y absolutas, y entenderá los componentes de ruta especiales `.` y `..`.

El comando `youngest` muestra la versión más joven. Lo cual es útil para determinar el rango válido de revisiones que puede usar como argumentos del comando `setrev`—se le permite navegar por todas las revisiones (le recordamos que son nombradas por números) entre 0 y la más joven inclusive. Determinar la transacción navegable válida no es tan bonito. Use el comando **lstxns** para obtener un listado de las transacciones que puede consultar. El listado de las transacciones navegables es el mismo listado que el devuelto por **svnadmin lstxns** y el mismo listado que es válido para ser usado con la opción `--transaction` de **svnlook**.

Una vez haya acabado de usar la línea de comando, puede salir de manera limpia usando el comando **exit**. Alternativamente, puede mandar un carácter de fin de línea—Control-D (aunque algunas distribuciones de Python para Win32 usan en su lugar la convención de Windows de Control-Z).

Utilidades de la base de datos Berkeley

Toda la estructura y datos versionados de su sistema de ficheros viven en un conjunto de tablas de la base de datos Berkeley dentro del subdirectorio `db` de su repositorio. Este subdirectorio es un directorio de entorno de base de datos Berkeley habitual, y por lo tanto puede ser usado con cualquiera de las herramientas de la base de datos Berkeley (puede ver su documentación en la página web de SleepyCat, `http://www.sleepycat.com/`).

Para un uso diario de Subversion, estas herramientas son innecesarias. La mayor parte de la funcionalidad típicamente requerida por los repositorios Subversion ha sido duplicada en la herramienta **svnadmin**. Por ejemplo, **svnadmin list-unused-dblogs** y **svnadmin list-dblogs** realizan un conjunto de lo que proporciona el comando **db_archive** de Berkeley, y **svnadmin recover** engloba los casos de uso habituales de la utilidad **db_recover**.

Aunque todavía hay algunas herramientas de la base de datos Berkeley que puede encontrar útiles. Los programas **db_dump** y **db_load** escriben y leen, respectivamente, un fichero de formato propio que describe las claves y valores de la base de datos Berkeley. Dado que las bases de datos Berkeley no son portables entre arquitecturas diferentes, este formato es útil para transferir las bases de datos de una máquina a otra, sin importar su arquitectura o sistema operativo. Además, la utilidad **db_stat** proporciona información útil sobre el estado de su entorno de base de datos Berkeley, incluyendo estadísticas detalladas sobre los bloqueos y subsistemas de almacenamiento.

Limpieza del repositorio

En general su repositorio Subversion requerirá poca atención una vez lo haya configurado a su gusto. No obstante, hay ocasiones en las que podría necesitar la asistencia manual de un administrador. La utilidad **svnadmin** proporciona algunas funcionalidades provechosas para realizar tareas como

- modificar mensajes de informes de cambios,
- eliminar transacciones muertas,

- recuperar repositorios “tocados”, y
- migrar el contenido de un repositorio a otro.

Quizás el subcomando de **svnadmin** más usado sea `setlog`. Cuando una transacción es guardada en el repositorio y ascendida a categoría de revisión, el mensaje del informe de cambios asociado con esa nueva revisión (y proporcionado por el usuario) se almacena como una propiedad no versionada adjunta a la propia revisión. En otras palabras, el repositorio sólo recuerda el último valor de la propiedad, y descarta valores anteriores.

A veces el usuario cometerá errores (ortográficos o quizás información equivocada) en sus informes de cambios. Si configura el repositorio (usando los ganchos `pre-revprop-change` y `post-revprop-change`; vea “Scripts de enganche”) para permitir modificaciones sobre el mensaje de informe de cambios de revisiones existentes, entonces el usuario puede “corregir” su informe remotamente usando el comando `propset` del programa **svn** (vea Capítulo 9, *Referencia completa de Subversion*). No obstante, debido al peligro potencial de perder información para siempre, los repositorios Subversion se configuran por defecto para impedir cambios a propiedades no versionadas—excepto para el administrador.

Si un informe de cambios necesita ser cambiado por un administrador, puede ejecutar **svnadmin setlog**. Éste comando modifica el mensaje de informe de cambios (la propiedad `svn:log`) de una revisión concreta de un repositorio, leyendo el nuevo valor del fichero proporcionado.

```
$ echo "Here is the new, correct log message" > newlog.txt
$ svnadmin setlog myrepos newlog.txt -r 388
```

El comando **svnadmin setlog** a secas sigue estando limitado por las mismas protecciones que impiden a los usuarios remotos modificar propiedades no versionadas—los ganchos `pre-` y `post-revprop-change` siguen ejecutándose, y por lo tanto debe configurar el repositorio para permitir cambios de esta naturaleza. Pero un administrador puede saltarse estas protecciones pasando el parámetro `--bypass-hooks` al comando **svnadmin setlog**.

Recuerde, no obstante, que al ignorar los ganchos, probablemente esté evitando acciones como notificaciones automáticas por correo informando de cambios a propiedades, sistemas de copias de seguridad que siguen los cambios de propiedades sin versionar, etc, etc. En otras palabras, sea muy cuidadoso con lo que vaya a cambiar, y cómo lo cambia.

Otro uso habitual de **svnadmin** es comprobar que en el repositorio no hay transacciones pendientes—posiblemente muertas. En el caso de que una transacción falle, ésta normalmente se anula. Es decir, la propia transacción y cualquier dato asociado con ella (y sólo con ella) es eliminada del repositorio. No obstante, ocasionalmente ocurre un fallo de tal modo que la limpieza de la transacción no se llega a realizar nunca. Esto podría pasar por varias razones: quizás la operación del cliente fue terminada de manera poco elegante por el usuario, quizás ocurrió un fallo de red durante la operación, etc. En cualquier caso, estas transacciones muertas sólo molestan en el repositorio y consumen recursos.

Puede usar el comando `lstxns` de **svnadmin**'s para obtener un listado de los nombres de las transacciones actuales de relevancia.

```
$ svnadmin lstxns myrepos
19
3a1
a45
$
```

Cada elemento del listado mostrado puede ser usado con **svnlook** (y su opción `--transaction`) para determinar quién creó la transacción, cuándo fue creada, qué tipos de cambios fueron realizados en la transacción—en otras palabras, si la transacción es o nó una candidata segura para ser eliminada. En caso de serlo, el nombre de la transacción puede ser usado con **svnadmin rmtxns**,

que realizará la limpieza de la transacción. ¡De hecho, el subcomando `rmtxns` puede usar directamente como entrada la salida de `lstxns`!

```
$ svnadmin rmtxns myrepos `svnadmin lstxns myrepos`
$
```

Si usa estos subcomandos de éste modo, debería considerar desactivar temporalmente el acceso al repositorio por parte de sus clientes. Así, nadie podrá comenzar una transacción legítima antes de que comience con la limpieza. Lo que viene a continuación es un poco de shell-scripting que puede generar rápidamente la información sobre cada transacción pendiente de su repositorio:

Ejemplo 5.2. txn-info.sh (Informe de transacciones pendientes)

```
#!/bin/sh

### Generate informational output for all outstanding transactions in
### a Subversion repository.

SVNADMIN=/usr/local/bin/svnadmin
SVNLOOK=/usr/local/bin/svnlook

REPOS="${1}"
if [ "x$REPOS" = x ] ; then
  echo "usage: $0 REPOS_PATH"
  exit
fi

for TXN in `${SVNADMIN} lstxns ${REPOS}`; do
  echo "---[ Transaction ${TXN} ]-------------------------------------------"
  ${SVNLOOK} info "${REPOS}" --transaction "${TXN}"
done
```

Puede ejecutar el script anterior usando **/ruta/a/txn-info.sh /ruta/al/repositorio**. La salida es básicamente la concatenación de varios bloques de salida de **svnlook info** (vea “svnlook”), y tendrá el siguiente aspecto:

```
$ txn-info.sh myrepos
---[ Transaction 19 ]-------------------------------------------
sally
2001-09-04 11:57:19 -0500 (Tue, 04 Sep 2001)
0
---[ Transaction 3a1 ]-------------------------------------------
harry
2001-09-10 16:50:30 -0500 (Mon, 10 Sep 2001)
39
Trying to commit over a faulty network.
---[ Transaction a45 ]-------------------------------------------
sally
2001-09-12 11:09:28 -0500 (Wed, 12 Sep 2001)
0
$
```

Habitualmente, si descubre una transacción muerta que no tiene adjunto un informe de cambios, se trata del resultado de una operación fallida de actualización (o de estilo similar). Estas operaciones usan transacciones de Subversion internamente para emular el estado de una copia de trabajo local. Dado que éstas transacciones no tienen como propósito ser almacenadas, Subversion no requiere que tengan un mensaje de informe de cambios. Las transacciones que tienen informes de cambios son casi con total seguri-

dad algún tipo de operación de almacenamiento fallida. Además, la fecha de una transacción puede proporcionar información interesante—por ejemplo, ¿qué probabilidades hay de que una operación que comenzase hace nueve meses todavía esté activa?

En resumen, no debe tomar a la ligera las decisión de limpieza de transacciones. Puede emplear varias fuentes de información—incluyendo los registros de acceso y errores de Apache, los registros de transacciones Subversion completadas con éxito, etc— durante la toma de decisiones. Finalmente, un administrador siempre puede comunicarse con el autor de una transacción aparentemente muerta (vía email, por ejemplo) para verificar que la transacción está, de hecho, en un estado zombi.

Gestionando el espacio de almacenamiento

Mientras que el coste del almacenamiento ha caído de manera increíble durante los últimos años, el espacio en disco sigue siendo un motivo de preocupación válido para los administradores que buscan versionar grandes cantidades de datos. Cada byte adicional consumido por el repositorio en vivo es un byte que debe ser almacenado en una copia de seguridad, quizás múltiples veces como parte de una política de rotado periódico de copias de seguridad. Dado que el principal medio de almacenamiento de un repositorio de Subversion es un complejo sistema de bases de datos, es útil saber qué piezas deben permanecer en el servidor activo, cuáles deben ser almacenadas en una copia de seguridad, y cuáles pueden ser eliminadas.

Hasta hace poco, el mayor culpable en cuanto al uso de disco duro por parte de un repositorio de Subversion eran los fichero de registro que usaba la base de datos Berkeley para realizar sus pre-escrituras antes de modificar realmente los ficheros de la base de datos. Éstos ficheros capturan todas las acciones tomadas durante la ruta que modifica el estado de la base de datos— si los ficheros de la base de datos reflejan en cualquier momento un estado determinado, los ficheros de registro contienen todos los cambios entre diferentes estados. Como tales, pueden comenzar a acumularse rápidamente.

Afortunadamente, y comenzando con la versión 4.2 de la base de datos Berkeley, el entorno de la base de datos tiene la capacidad para borrar sus propios ficheros de registro obsoletos sin necesidad de procedimientos externos. Cualquier repositorio creado usando **svnadmin**, el cual se compila contra la versión 4.2 o superior de la base de datos Berkeley, será configurado automáticamente para borrar sus ficheros de registros. Si no quiere activar esta característica, simplemente pase la opción `--bdb-log-keep` al comando **svnadmin create**. Si olvida hacer esto, o cambia de opinión posteriormente, puede editar el fichero `DB_CONFIG` ubicado en el directorio `db` de su repositorio, comentar la línea que contiene la directiva `set_flags DB_LOG_AUTOREMOVE`, y entonces ejecutar **svnadmin recover** sobre su repositorio para forzar los cambios de configuración. Vea “Configuración de la base de datos Berkeley” para más información sobre la configuración de la base de datos.

Sin mecanismo de algún tipo que elimine los ficheros de registro, éstos se acumularán a medida que use el repositorio. Esto es en cierto sentido una característica del sistema de base de datos—debería ser capaz de recrear su base de datos por completo sin usar nada más que los ficheros de registro, así que éstos pueden ser útiles en recuperaciones de la base de datos tras una catástrofe. Pero habitualmente querrá archivar los ficheros de registro que la base de datos Berkeley ya no usa, y entonces eliminarlos del disco para conservar espacio. Use el comando **svnadmin list-unused-dblogs** para obtener un listado de los ficheros de registro no usados:

```
$ svnadmin list-unused-dblogs /path/to/repos
/path/to/repos/log.0000000031
/path/to/repos/log.0000000032
/path/to/repos/log.0000000033

$ svnadmin list-unused-dblogs /path/to/repos | xargs rm
## disk space reclaimed!
```

Para mantener el tamaño del repositorio tan pequeño como sea posible, Subversion usa la *deltificación* (o bien, “almacenamiento deltificado”) dentro del propio repositorio. La deltificación conlleva codificar la representación de un bloque de datos como una colección de diferencias contra otro bloque de datos. Si dos piezas son muy similares, la deltificación proporcionará ganancias de almacenamiento para los bloques deltificados—en lugar de tomar tanto espacio como el tamaño de los datos originales, tomará justo lo necesario para decir, “tengo casi el mismo aspecto de este otro bloque de datos de ahí, excepto por las siguientes diferencias.” De manera más específica, cada vez que una nueva versión de un fichero es enviada al repositorio, Subversion codifica la versión anterior (en realidad, varias versiones anteriores) como un delta contra la nueva versión. El resultado es que la mayor parte de los datos que suelen ocupar espacio—en concreto, los contenidos de los ficheros versionados—se almacenan con un tamaño mucho menor que la representación original completa (“fulltext”) de esos datos.

Dado que todos los datos del repositorio Subversion sujetos a la deltificación se almacenan en un único fichero de base de datos Berkeley, reducir el tamaño de los valores almacenados no reducirá necesariamente el tamaño de éste fichero. La base de datos Berkeley no obstante mantendrá un registro interno de áreas internas sin usar de la base de datos, y usará esas áreas primero antes de volver a redimensionar el tamaño del fichero de la base de datos. Así que aunque la deltificación no produce ganancias de espacio inmediatas, puede enlentecer drásticamente el crecimiento futuro de la base de datos.

Restauración del repositorio

Para poder proteger sus datos en el repositorio, el motor de la base de datos usa un mecanismo de bloqueo. Éste se asegura de que no hay porciones de la base de datos modificadas simultáneamente por múltiples clientes de la base de datos, y que cada proceso ve los datos en un estado correcto cuando éstos son leídos. Cuando un proceso necesita cambiar algo en la base de datos, primero comprueba la existencia de un bloqueo en los datos objetivo. Si los datos no están bloqueados, el proceso los bloquea, hace los cambios que desea realizar, y desbloquea los datos. Esto obliga a otros procesos a esperar hasta que se elimine el bloqueo antes de que se les permita continuar accediendo a esa sección de la base de datos.

A lo largo del uso de su repositorio Subversion, los errores fatales (como quedarse sin espacio en disco duro o sin memoria) o las interrupciones pueden evitar que un proceso tenga la oportunidad de eliminar los bloqueos que inició sobre la base de datos. El resultado es que el motor de la base de datos del sistema se “wedged”. Cuando esto ocurre, cualquier intento de acceso al repositorio se bloqueará de manera indefinida (dado que cada nuevo cliente estará esperando a que los bloqueos desaparezcan—lo cual no va a ocurrir).

Primero, si esto ocurre en su repositorio, no se preocupe. El sistema de ficheros de Subversion tiene la ventaja de usar transacciones, puntos de control y ficheros de registro pre-escritura para asegurarse de que sólo los eventos más catastróficos [4] pueden destruir de manera permanente el entorno de la base de datos. Un administrador de repositorio suficientemente paranoico estará haciendo copias de seguridad del repositorio en otra máquina separada de alguna manera, pero todavía no es el momento de llamarle para que realice la restauración de una cinta.

En segundo lugar, use la siguiente receta para intentar “unwedge” su repositorio:

1. Asegúrese de que no hay procesos accediendo (o intentando acceder) al repositorio. Para repositorios en red, esto también significa tener que desconectar el servidor HTTP Apache.

2. Conviértase en el usuario que posee y gestiona el repositorio. Esto es importante, dado que recuperar el repositorio poseyendo el usuario erróneo puede alterar los permisos de los ficheros del repositorio de tal modo que será inaccesible incluso después de haberlo “unwedged”.

3. Ejecute el comando **svnadmin recover /ruta/al/repositorio**. Debería ver algo como esto:

```
Please wait; recovering the repository may take some time...

Recovery completed.
The latest repos revision is 19.
```

   Este comando puede tardar muchos minutos en completar su tarea.

4. Rearranque el servidor Subversion.

[4] Ej: disco duro + enorme electroimán = desastre.

Este procedimiento corrige casi todos los casos de bloqueos del repositorio. Asegúrese de ejecutar este comando como el usuario a quien le pertenece la base de datos y la gestiona, no basta con ser `root`. Parte del proceso de recuperación puede conllevar la regeneración de varios ficheros de bases de datos (por ejemplo, regiones de memoria compartidas). Al realizar la recuperación como `root`, éstos ficheros tendrán sus permisos, lo cual significa que incluso cuando recupere la conexión con el repositorio, los usuarios normales no serán capaces de acceder a él.

Si el procedimiento anterior, por alguna razón, no consigue desbloquear su repositorio, debería hacer dos cosas. En primer lugar, mueva su repositorio estropeado a otro lugar y recupere la última copia de seguridad que tenga del mismo. Entonces, envíe un correo electrónico a la lista de usuarios de Subversion (en `<users@subversion.tigris.org>`) describiendo su problema con todo detalle. Para los desarrolladores de Subversion mantener la integridad de los datos es una prioridad extremadamente alta.

Migrando un repositorio

Un sistema de ficheros de Subversion almacena sus datos repartidos por varias tablas de bases de datos de una forma generalmente comprensible (y sólo de interés) para los propios desarrolladores de Subversion. No obstante, pueden darse las circunstancias adecuadas que requieran que todos los datos, o un subconjunto de los mismos, sean recopilados en un fichero único, portable, y de formato simple. Subversion proporciona tal mecanismo implementado como la pareja de subcomandos de **svnadmin**: `dump` y `load`.

La razón más común para volcar y recargar un repositorio de Subversion es por cambios en el propio Subversion. A medida que Subversion madura, hay momentos en los que hace falta realizar cambios al esquema del motor de la base de datos que hacen a Subversion incompatible con versiones anteriores de su repositorio. La acción recomendada para actualizarse atravesando una de estas barreras de incompatiblidad es relativamente simple:

1. Usando su *actual* versión de **svnadmin**, vuelque su repositorio a ficheros de volcado.

2. Actualícese a la nueva versión de Subversion.

3. Mueva sus viejos repositorios a otra parte, y cree nuevos repositorios vacíos en su lugar usando su *nuevo* **svnadmin**.

4. Otra vez con su **svnadmin** *nuevo*, cargue sus ficheros de volcado en sus repositorios recién creados respectivamente.

5. Finalmente, asegúrese de copiar en los nuevos repositorios cualquier personalización realizada en los antiguos, incluyendo los ficheros `DB_CONFIG` y los ficheros de ganchos. Le recomendamos que preste atención a las notas de lanzamiento de la nueva versión de Subversion para comprobar si algún cambio desde su última actualización afecta a esos ficheros de ganchos u opciones de configuración.

svnadmin dump generará en su salida un rango de revisiones del repositorio usando el formato propio de Subversion para volcar el sistema de ficheros. El volcado formateado será enviado al flujo estándar de salida, mientras que los mensajes informativos son enviados al flujo estándar de errores. Esto le permite redirigir el flujo de salida estándar a un fichero mientras observa el estado del volcado en su terminal. Por ejemplo:

```
$ svnlook youngest myrepos
26
$ svnadmin dump myrepos > dumpfile
* Dumped revision 0.
* Dumped revision 1.
* Dumped revision 2.
…
* Dumped revision 25.
* Dumped revision 26.
```

Al final del proceso tendrá un único fichero (`dumpfile` según el ejemplo anterior) que contendrá todos los datos almacenados en su repositorio en el rango de revisiones especificado. Tenga en cuenta que **svnadmin dump** lee los árboles de versiones del repositorio igual que cualquier otro proceso "lector" (por ejemplo, **svn checkout**). Así que es seguro ejecutar este comando en cualquier momento.

El otro subcomando de la pareja, **svnadmin load**, procesa el flujo estándar de entrada como un fichero de volcado de repositorio de Subversion, y reproduce de manera efectiva esas revisiones volcadas en el repositorio destino durante la operación. También proporciona mensajes informativos, esta vez usando el flujo de salida estándar:

```
$ svnadmin load newrepos < dumpfile
<<< Started new txn, based on original revision 1
     * adding path : A ... done.
     * adding path : A/B ... done.
     …
------- Committed new rev 1 (loaded from original rev 1) >>>

<<< Started new txn, based on original revision 2
     * editing path : A/mu ... done.
     * editing path : A/D/G/rho ... done.

------- Committed new rev 2 (loaded from original rev 2) >>>

…

<<< Started new txn, based on original revision 25
     * editing path : A/D/gamma ... done.

------- Committed new rev 25 (loaded from original rev 25) >>>

<<< Started new txn, based on original revision 26
     * adding path : A/Z/zeta ... done.
     * editing path : A/mu ... done.

------- Committed new rev 26 (loaded from original rev 26) >>>
```

Dado que **svnadmin** usa la los flujos estándar de entrada y salida para el volcado de repositorio y el proceso de carga, las personas que se sientan atrevidas pueden probar algo como ésto (quizás usando diferentes versiones de **svnadmin** en cada lado de la tubería):

```
$ svnadmin create newrepos
$ svnadmin dump myrepos | svnadmin load newrepos
```

Hemos mencionado anteriormente que **svnadmin dump** genera un rango de revisiones. Use la opción `--revision` para indicar que quiere volcar una revisión concreta, o un rango. Si omite esta opción, todas las revisiones existentes del repositorio serán volcadas.

```
$ svnadmin dump myrepos --revision 23 > rev-23.dumpfile
$ svnadmin dump myrepos --revision 100:200 > revs-100-200.dumpfile
```

A medida que Subversion vuelca cada revisión nueva, muestra la información justa para permitir a un cargador futuro recrear esa revisión basándose en la anterior. En otras palabras, para cualquier revisión dada en el fichero de volcado, sólo aparecerán los elementos que cambiaron en ella. La única excepción a esta regla es la primera revisión volcada por el comando **svnadmin dump** actual.

Por defecto, Subversion no expresará la primera revisión volcada como las meras diferencias aplicables sobre la revisión anterior.

Para empezar, ¡es que no hay revisión anterior alguna en el fichero de volcado! Y en segundo lugar, Subversion no puede saber el estado del repositorio en el cual los datos volcados serán cargados (si este hecho llega a producirse). Para asegurarse de que la salida de cada ejecución de **svnadmin dump** es autosuficiente, la primera revisión volcada es por defecto una representación completa de cada directorio, fichero, y propiedad en esa revisión del repositorio.

No obstante, puede cambiar este comportamiento por defecto. Si añade la opción `--incremental` cuando vuelque su repositorio, **svnadmin** comparará la primera versión volcada del repositorio contra la revisión anterior del repositorio, igual que trata cualquier otra revisión volcada. Entonces mostrará la primera revisión exactamente igual que el resto de las revisiones del rango de volcado—mencionando únicamente los cambios ocurridos en esa revisión. El beneficio de esto es que puede crear varios ficheros de volcado menores que pueden ser cargados secuencialmente, en lugar de un único fichero grande, siguiendo estos pasos:

```
$ svnadmin dump myrepos --revision 0:1000 > dumpfile1
$ svnadmin dump myrepos --revision 1001:2000 --incremental > dumpfile2
$ svnadmin dump myrepos --revision 2001:3000 --incremental > dumpfile3
```

Estos ficheros de volcado podrían ser cargados en un nuevo repositorio usando la siguiente secuencia de comandos:

```
$ svnadmin load newrepos < dumpfile1
$ svnadmin load newrepos < dumpfile2

$ svnadmin load newrepos < dumpfile3
```

Otro truco interesante que puede realizar con la opción `--incremental` consiste en añadir a un fichero de volcado existente un nuevo rango de revisiones volcadas. Por ejemplo, digamos que tiene un gancho `post-commit` que simplemente añade el volcado del repositorio de la versión única que despertó la ejecución del gancho. O quizás tenga un script que se ejecute por las noches para añadir los datos del fichero de volcado de todas las revisiones que fueron añadidas al repositorio desde la última vez que fue ejecutado el script. Usados de este modo, los comandos **svnadmin** `dump` y `load` pueden ser valiosos aliados para realizar copias de seguridad de su repositorio a lo largo del tiempo en caso de un fallo de sistema o algún otro evento catastrófico.

El formato de volcado también puede ser usado para fusionar los contenidos de varios repositorios diferentes en uno único. Al usar la opción `--parent-dir` de **svnadmin load**, puede especificar un nuevo directorio virtual para el proceso de carga. Lo cual significa que si tiene ficheros de volcados para tres repositorios, digamos `calc-dumpfile`, `cal-dumpfile`, y `ss-dumpfile`, puede crear primero un nuevo repositorio para almacenarlos todos:

```
$ svnadmin create /path/to/projects
$
```

Entonces, cree nuevos directorios en el repositorio que encapsularán los contenidos de cada uno de los tres repositorios anteriores:

```
$ svn mkdir -m "Initial project roots" \
      file:///path/to/projects/calc \
      file:///path/to/projects/calendar \
      file:///path/to/projects/spreadsheet
Committed revision 1.
$
```

Por último, cargue los ficheros de volcado individuales en sus ubicaciones respectivas del nuevo repositorio:

```
$ svnadmin load /path/to/projects --parent-dir calc < calc-dumpfile
…
$ svnadmin load /path/to/projects --parent-dir calendar < cal-dumpfile
…
```

```
$ svnadmin load /path/to/projects --parent-dir spreadsheet < ss-dumpfile
…
$
```

Mencionaremos un último uso del formato de volcado de repositorio de Subversion—conversión desde un formato de almacenamiento diferente o sistema de control de versiones. Dado que el fichero de volcado es, en su mayor parte, legible por un humano, [5] debería ser relativamente sencillo describir un conjunto genérico de cambios—cada uno de ellos debería ser descrito como una nueva revisión—usando este formato de fichero. De hecho, la utilidad **cvs2svn.py** (vea “Convirtiendo un repositorio de CVS a Subversion”) usa el formato de volcado para representar el contenido de un repositorio CVS para que sus contenidos pueden ser trasladados a un repositorio Subversion.

Copias de seguridad del repositorio

A pesar de numerosos avances tecnológicos desde el nacimiento de la computadora moderna, hay una cosa que se entiende con cristalina claridad—a veces, las cosas van muy, pero que muy mal. Pérdidas de corriente, fallos de conectividad en la red, RAM corrupta y discos duros estropeados son un mero avance del mal que el Destino puede desatar sobre incluso el administrador más concienzudo. Y por lo tanto, llegamos a un tema muy importante—cómo hacer copias de seguridad de los datos de su repositorio.

En general hay dos tipos de métodos de copia de seguridad disponibles para los administradores de repositorios de Subversion—incrementales y completos. Hemos discutido en una sección anterior de este capítulo cómo usar **svnadmin dump --incremental** para realizar copias de seguridad incrementales (vea “Migrando un repositorio”). Esencialmente, la idea es realizar en un momento dado copias de seguridad de los cambios realizados desde la última vez que se hizo la copia de seguridad anterior.

Una copia de seguridad completa del repositorio es casi una copia literal del directorio que contiene todo el repositorio (lo cual incluye el entorno de la base de datos Berkeley). Ahora, a no ser que desactive temporalmente todo el acceso a su repositorio, ejecutar un simple copiado recursivo del directorio conlleva el riesgo de generar una copia de seguridad errónea, dado que alguien podría estar modificando la base de datos.

Afortunadamente, los documentos de la base de datos Berkeley de Sleepycat describen un orden concreto en el cual se pueden copiar los ficheros de la base de datos de tal modo que se garantice una copia de seguridad válida. Y aun mejor, usted no necesita implementar ese algoritmo por su cuenta, porque el equipo de desarrolladores de Subversion ya lo ha hecho. El script **hot-backup.py** que puede encontrar en el directorio `tools/backup/` de la distribución de código fuente de Subversion. Usando como parámetros la ruta al repositorio y a la ubicación de la copia de seguridad, **hot-backup.py**—que realmente no es más que un envoltorio más inteligente sobre el comando **svnadmin hotcopy**— realizará los pasos necesarios para realizar una copia de seguridad en caliente del repositorio—sin necesitar en absoluto tener que impedir el acceso al mismo—y después borrará los ficheros de registro sin usar de Berkeley de su repositorio activo actual.

Incluso si también dispone de una copia de seguridad incremental, quizás quiera ejecutar también este programa de manera habitual. Por ejemplo, podría considerar añadir **hot-backup.py** a un programa de planificación (como por ejemplo **cron** en sistemas Unix). O, si prefiere soluciones de copias de seguridad con alta granularidad, podría hacer que su gancho post-commit llame **hot-backup.py** (vea “Scripts de enganche”), lo cual provocará una nueva copia de seguridad de su repositorio con cada nueva versión. Simplemente añada el script `hooks/post-commit` a su directorio activo del repositorio:

```
(cd /path/to/hook/scripts; ./hot-backup.py ${REPOS} /path/to/backups &)
```

La copia de seguridad resultante es un repositorio de Subversion completamente funcional, capaz de reemplazas su repositorio activo en caso de que algo vaya muy mal.

Ambos métodos de copias de seguridad son beneficiosos. La copia de seguridad completa es de lejos la más sencilla, la que siempre resultará ser una réplica de su repositorio perfectamente funcional. Le recordamos que esto significa que si algo malo le ocurre a su repositorio activo, puede recuperar su copia de seguridad con un simple copiado recursivo de directorios. Desafortunadamente, si está manteniendo múltiples copias de seguridad de su repositorio, estas copias completas acabarán ocupando la misma cantidad de espacio de disco que su repositorio activo.

[5] El formato de volcado de repositorio de Subversion recuerda al formato RFC-822, el mismo tipo de formato usado para la mayor parte del correo electrónico.

Las copias de seguridad incrementales usando el formato de volcado del repositorio son excelentes para tener a mano si el esquema de la base de datos cambia entre sucesivas versiones del propio Subversion. Dado que para actualizar su repositorio al nuevo esquema es necesario que haga un volcado y carga completa del repositorio, es muy conveniente tener la mitad de ese proceso (la parte del volcado) realizada. Desafortunadamente, la creación—y recuperación—de copias de seguridad incrementales tarda mucho más, dado que cada revisión es reproducida en el fichero de volcado o el repositorio.

En ambos escenarios, los administradores del repositorio necesitan estar atentos a cómo las modificaciones de propiedades de revisión no versionadas afectan a sus copias de seguridad. Dado que estos cambios no generan por sí mismos nuevas revisiones, no activarán ganchos post-commit, y quizás ni siquiera los ganchos pre-revprop-change y post-revprop-change. [6] Y dado que puede cambiar las propiedades de revisión sin respetar su orden cronológico—puede cambiar cualquier propiedad de revisión en cualquier momento—una copia de seguridad incremental de las últimas pocas revisiones quizás no capture la modificación de una propiedad de revisión que fue incluida como parte de una copia de seguridad anterior.

Hablando de manera general, sólo los realmente paranoicos necesitarían hacer una copia de seguridad de su repositorio completo, digamos que, cada vez que se realiza un cambio. No obstante, asumiendo que un repositorio determinado tiene funcionando otros mecanismos de redundancia con relativa granularidad (como correos electrónicos por modificación), una copia de seguridad en caliente de la base de datos sería algo que un administrador de repositorio querría incluir como parte de una copia de seguridad de sistema nocturna. Para la mayoría de los repositorios, los correos archivados con cambios son suficiente fuente de redundancia, al menos para los últimos cambios. Pero son sus datos—protéjalos tanto como desee.

A menudo, la mejor estrategia para realizar copias de seguridad es aquella que diversifique. Puede aprovechar combinaciones de copias de seguridad completas e incrementales, y adicionalmente archivos de correo con los cambios. Los desarrolladores de Subversion, por ejemplo, realizan una copia de seguridad del repositorio de código fuente tras cada nueva revisión, y mantienen un archivo de todas las notificaciones por correo electrónico de cambios y modificaciones de propiedades. Su solución podría ser similar, pero debería estar orientada a las necesidades y ese delicado balance entre la conveniencia y la paranoia. Y aunque nada de todo esto le ayude a salvar su hardware del puño de hierro del Destino, [7] ciertamente debería ayudarle a recuperarse tras uno de esos momentos.

Añadiendo proyectos

Una vez haya creado y configurado su repositorio, todo lo que queda es comenzar a usarlo. Si tiene una colección de datos existentes listos para ser puestos bajo control de versiones, es muy probable que desee usar el subcomando `import` del programa cliente **svn**. No obstante, antes de hacer esto debería considerar con cuidado los planes a largo plazo de su repositorio. En esta sección le ofreceremos algunos consejos sobre la planificación del esquema de su repositorio, y cómo reorganizar sus datos de acuerdo con éste.

Escogiendo el esquema de repositorio

Aunque Subversion le permite desplazar los ficheros y directorios versionados sin pérdida de información de cualquier tipo, al hacerlo puede entorpecer el flujo de trabajo de aquellos quienes acceden al repositorio con frecuencia y esperan encontrar algunas cosas en ubicaciones determinadas. Intente asomarse un poco al futuro; planifique antes de poner sus datos bajo control de versiones. Al “organizar” el contenido de sus repositorios de una manera efectiva la primera vez, podrá evitar un montón de futuros dolores de cabeza.

Hay algunas cosas que deberá considerar al configurar repositorios con Subversion. Asumamos que como administrador de repositorio será responsable de dar soporte del sistema de control de versiones a varios proyectos. La primera decisión consiste en decidir si usará un único repositorio para múltiples proyectos, o le dará un repositorio a cada proyecto, o una mezcla de cambos.

Hay múltiples beneficios en el uso de un único repositorio para múltiples proyectos, siendo el más obvio evitar duplicar el trabajo de mantenimiento. Tener un único repositorio significa que sólo hay un conjunto de scripts de enganche, un elemento del que hacer rutinariamente copias de seguridad, un único volcado y recarga si Subversion lanza alguna nueva versión incompatible, y así con

[6] **svnadmin setlog** puede ser invocado de un modo que evita por completo el mecanismo de ganchos.
[7] Ya sabe—el término genérico para sus “erráticos dedos”.

todo. Además, puede mover datos entre proyectos muy fácilmente, y sin perder ninguna información histórica de versionado.

Las desventajas de usar un único repositorio son que los diferentes proyectos pueden tener diferentes listas de correo que reciben notificaciones de los cambios realizados o diferentes requisitos de autenticación y autorización. Además, recuerde que Subversion usa números de revisión globales por repositorio. Hay gente a la que no le gusta el hecho de que incluso sin haber realizado cambios recientes en sus proyectos, el número de la revisión más reciente sigue subiendo porque otros proyectos siguen añadiendo revisiones de manera activa.

También puede diseñar una solución híbrida. Por ejemplo, los proyectos pueden ser agrupados en función de las relaciones que tengan entre ellos. Quizás tenga varios repositorios con un puñado de proyectos en cada uno. De ese modo, los proyectos más propensos a compartir sus datos pueden hacerlo de manera sencilla, y a medida que se añaden revisiones al repositorio, al menos los desarrolladores del mismo saben que están relacionadas remotamente con todo aquél que usa ese repositorio.

Tras decidir cómo desea organizar sus proyectos con respecto a los repositorios, probablemente querrá decidir las jerarquías de los directorios almacenados en ellos. Dado que Subversion usa copias de directorio normales para hacer ramas y etiquetas (vea Capítulo 4, *Crear ramas y fusionarlas*), la comunidad de Subversion recomienda que elija una ubicación en el repositorio para cada *raíz de proyecto*— el directorio “superior” que contiene todos los datos relacionados con el mismo—y cree tres nuevos subdirectorios en esa raíz: `trunk`, que sería el directorio bajo el cual se lleva el desarrollo principal del proyecto; `branches`, que sería el directorio que alojaría las diferentes ramas con nombre de la línea principal de desarrollo; `tags`, que es el directorio donde las ramas son creadas, y quizás destruidas, pero nunca modificadas.

Por ejemplo, su repositorio podría tener este aspecto:

```
/
   calc/
      trunk/
      tags/
      branches/
   calendar/
      trunk/
      tags/
      branches/
   spreadsheet/
      trunk/
      tags/
      branches/
   …
```

Tenga en cuenta que no importa dónde está la raíz de cada proyecto en su repositorio. Si sólo tiene un proyecto por repositorio, el lugar lógico para poner la raíz de cada proyecto es en la raíz del repositorio respectivo. Si tiene múltiples proyectos, quizás desee organizarlos en grupos dentro del repositorio, quizás ubicando proyectos con fines similares o código compartido en el mismo subdirectorio, o quizás simplemente prefiera agruparlos alfabéticamente. Tal esquema podría tener el siguiente aspecto:

```
/
   utils/
      calc/
         trunk/
         tags/
         branches/
      calendar/
         trunk/
         tags/
         branches/
      …
   office/
      spreadsheet/
         trunk/
         tags/
```

```
        branches/
    …
```

Despliegue su repositorio de la manera que crea más conveniente. Subversion no espera ni obliga un esquema de directorios concreto—a sus ojos, un directorio es un directorio es un directorio. Por último, debería elegir la organización del repositorio de manera que satisfaga las necesidades de aquellos que trabajen en los proyectos ahí alojados.

Creando el esquema, importando los datos iniciales

Tras decidir cómo organizar los proyectos de su repositorio, probablemente quiera poblarlo con ese esquema y los datos iniciales de los proyectos. Hay varios modos de hacer esto en Subversion. Podría usar el comando **svn mkdir** (vea Capítulo 9, *Referencia completa de Subversion*) para crear cada directorio en su esquema esquelético, uno tras otro. Una forma más rápida de realizar esta misma tarea es usar el comando **svn import** (vea "**svn import**"). Al crear el esquema en una ubicación temporal de su disco, puede importarlo por completo en un único cambio:

```
$ mkdir tmpdir
$ cd tmpdir
$ mkdir projectA
$ mkdir projectA/trunk
$ mkdir projectA/branches
$ mkdir projectA/tags
$ mkdir projectB
$ mkdir projectB/trunk
$ mkdir projectB/branches
$ mkdir projectB/tags
…
$ svn import . file:///path/to/repos --message 'Initial repository layout'
Adding         projectA
Adding         projectA/trunk
Adding         projectA/branches
Adding         projectA/tags
Adding         projectB
Adding         projectB/trunk
Adding         projectB/branches
Adding         projectB/tags
…
Committed revision 1.
$ cd ..
$ rm -rf tmpdir
$
```

Puede comprobar los resultados de la operación de importado ejecutando el comando **svn list**:

```
$ svn list --verbose file:///path/to/repos
      1 harry               May 08 21:48 projectA/
      1 harry               May 08 21:48 projectB/
…
$
```

Una vez tenga la estructura de su esquema en lugar, puede comenzar a importar los del proyecto en su repositorio, si es que tales datos existen. De nuevo, hay varios métodos para hacer esto. Puede usar el comando **svn import**. Podría obtener una copia de trabajo local de su reciente repositorio, mover y organizar los datos de su proyecto dentro, y usar los comandos **svn add** y **svn commit**. Pero una vez comenzamos a hablar de tales cosas, ya no estamos discutiendo la administración del repositorio. Si no está familiarizado con el programa cliente **svn**, vea Capítulo 3, *Recorrido guiado*.

Sumario

A estas alturas debería tener un conocimiento básico sobre cómo crear, configurar, y mantener repositorios de Subversion. Le hemos introducido a varias herramientas que le asistirán con estas tareas. A lo largo del capítulo hemos avisado sobre los posibles problemas, y proporcionado sugerencias para evitarlos.

Todo lo que queda es decidir qué datos excitantes almacenará en su repositorio, y finalmente, cómo hacerlos disponibles por red. El siguiente capítulo está dedicado a todo lo relacionado con la red.

Capítulo 6. Configuración del servidor

Un repositorio Subversion puede ser accedido simultáneamente por clientes que son ejecutados en la misma máquina en la que reside el repositorio usando el método `file:///`. Pero una típica configuración de Subversion implica a un servidor al que acceden clientes desde sus computadoras en el entorno de una oficina— o quizás, alrededor del mundo.

Esta sección describe como publicar su repositorio Subversion para ser accedido por clientes remotos. Cubriremos los tipos de mecanismos de servidor disponibles actualmente , debatiendo la configuración y uso de cada uno de éstos. Después de que haya leído está sección, usted debería estar capacitado para decidir que tipo de configuración de red es la apropiada para sus necesidades, y comprender como realizar la instalación en su computadora.

Introducción

Subversion cuenta con un diseño de la capa de red abstracto. Esto significa que el repositorio puede ser accedido por cualquier clase de proceso de servidor y el API[1] de "acceso al repositorio" cliente permite a los programadores escribir componentes que se comuniquen a través de su protocolo de red pertinente. En teoría, Subversion puede lucir una infinidad de implementaciones de red. En la práctica, sólo existen dos tipos de servidores al momento de escribirse este documento.

Apache es un servidor web extremadamente popular; utilizando el módulo **mod_dav_svn**, Apache puede acceder a un repositorio y hacerlo disponible a los clientes a través del protocolo WebDAV/DeltaV, el cuál es una extensión del HTTP. En la otra esquina se encuentra **svnserve**: un pequeño programa servidor independiente que se comunica a través de un protocolo personalizado con los clientes. La Tabla 6-1 presenta una comparación de ambos tipos de servidores.

Observe que Subversion, siendo un proyecto de código abierto, no apoya oficialmente a ningún servidor como opción "primaria" u "oficial". Ninguna de las implementaciones de red es considerada como un ciudadano de segunda clase; cada servidor posee ventajas y desventajas distintivas. De hecho, es posible ejecutar los diferentes servidores en paralelo, cada uno accediendo a sus repositorios a su modo y sin obstaculizar al otro (ver "Ofrecer múltiples métodos de acceso al repositorio"). A continuación una breve visión general y comparación de los servidores Subversion disponibles—es tiempo de que elija la instalación que sea mejor para usted y sus usuarios.

Tabla 6.1. Comparación de tipos de servidores de red

Característica	Apache + mod_dav_svn	svnserve
Opciones de autenticación	Autorización básica HTTP(S), Certificados X.509, LDAP, NTLM, o cualquier otro mecanismo disponible por Apache httpd	CRAM-MD5 o SSH
Opciones de cuenta de usuario	archivo privado de usuarios 'users'	archivo privado de usuarios 'users', o cuentas (SSH) existentes en el sistema
Opciones de autorización	acceso general de lectura/escritura, o control de acceso por directorios	acceso general de lectura/escritura
Cifrado	a través de SSL (opcional)	a través de un túnel SSH (opcional)
Interoperabilidad	disponible parcialmente por otros clientes WebDAV	no posee interoperabilidad
Ver a través de la web	soporte integrado limitado, o a través de herramientas de terceras partes tales como ViewCVS	a través de herramientas de terceras partes tales como ViewCVS
Velocidad	algo lento	algo rápido
Configuración inicial	algo compleja	bastante simple

[1] N.T.: Application programmmers interface, o interfaz de programación de aplicaciones. Este acrónimo se considera reconocido internacionalmente motivo por el cuál no será traducido.

Modelo de red

En esta sección se debate en términos generales como interactúan el cliente y servidor Subversión entre sí, independientemente de la implementación de red que esté utilizando. A su término, usted comprenderá como se comporta un servidor y las diferentes formas en que un cliente puede ser configurado para responder.

Solicitudes y Respuestas

El cliente Subversion insume la mayor parte de su tiempo manejando copias de trabajo. No obstante, cuando necesita información del repositorio efectúa una solicitud a través de la red, y el servidor responde apropiadamente. Los detalles del protocolo de red son ocultados por el usuario; el cliente intenta acceder a una URL, y dependiendo del esquema URL, utiliza un protocolo particular para contactar al servidor (ver URLs del repositorio). Los usuarios pueden ejecutar **svn --version** para ver que esquemas URL y protocolos que comprende el cliente.

Cuando el proceso del servidor recibe una solicitud del cliente, típicamente demanda que el cliente se identifique. Este solicita la autenticación al cliente , y el cliente responde proporcionando las *credenciales* al servidor. Una vez completado ésto, el servidor responde con la información que el cliente solicitó originalmente. Observe que este método se diferencia de sistemas como CVS, dónde el cliente por cuenta propia ofrece las credenciales (“de acceso al sistema”), antes de realizar una solicitud. En Subversion, el servidor “exige tomar” las credenciales a través de la negociación con el cliente en el momento apropiado, en lugar de que el cliente las “impulse”. Esto asegura operaciones más elegantes. Por ejemplo, si se configura un servidor para que permita acceso de lectura al repositorio a todo el mundo, nunca demandará a un cliente que se autentique cuando intente realizar un **svn checkout**.

Si la solicitud de red del cliente escribe datos nuevos en el repositorio (por ejemplo: **svn commit**), un nuevo árbol de revisión es creado. Si la solicitud del cliente fue autenticada, el nombre del usuario es guardado como valor de la propiedad `svn:author` en la nueva revisión (ver “Propiedades no versionadas”). Si el cliente no fue autenticado (en otras palabras, si el servidor nunca promulgó la autenticación), la propiedad `svn:author` de la revisión está en blanco.[2]

Client Credentials Caching

Muchos servidores son configurados para requerir autenticación por cada solicitud que reciban. Esto puede convertirse en una gran molestia para los usuarios, quienes son forzados a escribir sus contraseñas una y otra vez.

Afortunadamente, el cliente Subversion ha remediado ésto: un sistema incorporado para ocultar las credenciales en el disco. Por defecto, en el momento en que el cliente se autentica satisfactoriamente con un servidor, éste guarda las credenciales en el área de configuración de ejecución privada del usuario— en `~/.subversion/auth/` en los sistemas tipo Unix o `%APPDATA%/Subversion/auth/` en Windows. (El área de ejecución es cubierto con mayor detalle en “Área de configuración de parámetros de ejecución”.) Con éxito las credenciales son ocultadas en el disco, encerradas en una combinación de hostname, puerto y realm de autenticación.

Cuando el cliente recibe la solicitud para que se autentique, primero busca las credenciales apropiadas en el caché de disco; sino existen, o si las credenciales fallan en la autenticación, el cliente simplemente pregunta al usuario por la información.

Las personas que sufren de paranoia sobre asuntos de seguridad pueden estar pensando, “Contraseñas ocultas en el disco? Eso es terrible!” Pero por favor conserve la calma. Primero, el área de caché `auth/` se encuentra protegida por permisos a fin de que sólo el usuario (dueño) puede leer la información que se encuentra en ella, evitando que otros puedan acceder. Si eso no es lo suficientemente seguro para usted, puede deshabilitar el caché de credenciales. Para lograrlo, sólo basta con un único comando, pasando la opción `--no-auth-cache`:

```
$ svn commit -F log_msg.txt --no-auth-cache
Authentication realm: <svn://host.example.com:3690> example realm
Username:  joe
Password for 'joe':

Adding         newfile
Transmitting file data .
```

[2]Este problema es actualmente una FAQ, como resultado de una configuración equivocada de la instalación.

```
Committed revision 2324.

# password was not cached, so a second commit still prompts us

$ svn rm newfile
$ svn commit -F new_msg.txt
Authentication realm: <svn://host.example.com:3690> example realm
Username:  joe
[...]
```

O, si usted desea deshabilitar el caché de credenciales en forma permanente, puede editar su archivo de configuración `config` (ubicado dentro del directorio `auth/`). Simplemente coloque `no` a `store-auth-creds`, y las credenciales no serán guardadas en el caché nunca más.

```
[auth]
store-auth-creds = no
```

Algunas veces los usuario desearán eliminar credenciales específicas del caché de disco. Para hacer ésto, necesitará navegar dentro del área `auth/` y eliminar el archivo de caché apropiado en forma manual. Las credenciales son almacenadas en el área de caché en archivos individuales; si usted mira dentro de cada archivo, verá claves y valores. La clave `svn:realmstreng` describe el "realm" servidor particular con el que que el archivo es asociado:

```
$ ls ~/.subversion/auth/svn.simple/
5671adf2865e267db74f09ba6f872c28
3893ed123b39500bca8a0b382839198e
5c3c22968347b390f349ff340196ed39

$ cat ~/.subversion/auth/svn.simple/5671adf2865e267db74f09ba6f872c28

K 8
username
V 3
joe
K 8
password
V 4
blah
K 15
svn:realmstring
V 45
<https://svn.domain.com:443> Joe's repository
END
```

Una vez que haya localizado el archivo de caché apropiado, sencillamente elimínelo.

Una última palabra acerca del comportamiento de autenticación del cliente: es necesaria una pequeña explicación respecto de las opciones `--username` y `--pasword`. Muchos sub-comandos del cliente aceptan estas opciones; sin embargo, es importante comprender que la utilización de las mismas *no* envía automáticamente las credenciales al servidor. Como se dijo anteriormente, el servidor “exige tomar” las credenciales del cliente cuando lo considera necesario; el cliente no puede “impulsarlas” a voluntad. Si un usuario y/o contraseña son pasados como opción, ellos serán presentados al servidor *sólo* si este así lo demanda. [3] Típicamente, estas opciones son utilizadas cuando:

[3]Una vez más, un error frecuente es configurar de forma equivocada el servidor para que este nunca emita una demanda de autenticación. Cuando los usuarios pasen la opción `--username` y `--pasword` al cliente, ellos se sorprenderán al ver que éstos nunca son utilizados, es decir, a pesar de aparecer una nueva revisión, ha sido enviada anónimamente!

- el usuario desea autenticarse con un nombre diferente a su usuario de sistema, o
- un guión desea autenticarse sin utilizar las credenciales del caché.

Aquí hay un resumen final que describe como se comporta un cliente Subversion cuando recibe la demanda de autenticación:

1. Verifica si el usuario ha especificado algunas credenciales como opción de línea de comando, a través de la opción `--username` y/o `--pasword`. Sino, o si esas opciones no pertenecen a una autenticación válida, entonces
2. Busca en el "realm" del servidor en el área de ejecución `auth/`, para ver si el usuario ya dispone de credenciales apropiadas en el caché. Sino, o si las credenciales del caché fallan en la autenticación, entonces
3. Recurre a preguntarle al usuario.

Si el cliente se autentica correctamente a través de alguno de los métodos listados aquí arriba, éste intentará guardar en las credenciales en el caché de disco (a no ser que el usuario haya deshabilitado ese funcionamiento, como hemos mencionado anteriormente.)

svnserve, un servidor personalizado

El programa **svnserve** es un servidor ligero, capaz de hablar a clientes sobre TCP/IP utilizando un protocolo con estado personalizado. Los clientes contactan un servidor **svnserve** utilizando URLs que comienzan con el esquema `svn://` o `svn+ssh://`. Esta sección explica las diferentes maneras de ejecutar un **svnserve**, cómo los clientes se autentican ellos mismos con el servidor, y cómo se configura un control de acceso apropiado a su repositorio.

Invocando el Servidor

Existen unas cuantas maneras diferentes de invocar el programa **svnserve**. Si se lo invoca sin argumentos, usted no verá nada más que un mensaje de ayuda. No obstante, si planea que **inetd** dispare el proceso, puede pasar la opción `-i` (`--inetd`):

```
$ svnserve -i
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
```

Cuando es invocado con la opción `--inetd`, **svnserve** intenta comunicarse con un cliente Subversion a través de *stdin* y *stdout* utilizando un protocolo personalizado. Este es un comportamiento estándar de los programas que son ejecutados a través de **inetd**. El IANA ha reservado el puerto 3690 para el protocolo Subversion, por lo tanto en un sistema tipo Unix usted puede agregar líneas a `/etc/services` como éstas (si es que no existen todavía):

```
svn             3690/tcp   # Subversion
svn             3690/udp   # Subversion
```

Y si su sistema utiliza un demonio **inetd** tipo Unix clásico, puede agregar esta línea a `/etc/inetd.conf`:

```
svn stream tcp nowait svnowner /usr/local/bin/svnserve svnserve -i
```

Asegúrese que “svnowner” es un usuario que posee permisos apropiados para acceder a su repositorio. Ahora, cuando una conexión cliente ingrese a su servidor en el puerto 3690, **inetd** disparará un proceso **svnserve** para servirlo.

Una segunda opción es ejecutar **svnserve** un proceso “demonio” independiente. Para esto, utilice la opción `-d`:

```
$ svnserve -d
$              # svnserve is now running, listening on port 3690
```

Cuando **svnserve** ejecute en modo demonio, usted puede utilizar las opciones `--listen-port=` y `--listen-host=` para personalizar el puerto y nombre exacto en el cual “escuchará”.

Aún queda una tercera forma de invocar a **svnserve**, y esa es en “modo túnel”, con la opción `-t`. Este modo supone que un programa de servicio remoto tal como **RSH** o **SSH** ha autenticado un cliente exitosamente y ahora está invocando un proceso privado **svnserve** *como tal usuario*. El programa **svnserve** se comporta normalmente (comunicándose a través de *stdin* y *stdout*), y asume que el tráfico está siendo redirigido automáticamente a través de algún tipo de túnel de vuelta al cliente. Cuando **svnserve** es invocado por un agente túnel como éste, está seguro que el usuario autenticado posee acceso total de lectura y escritura a los archivos de la base de datos del repositorio. (Ver Servidores y Permisos: Una Palabra de Advertencia.) Esencialmente es lo mismo que un usuario local accediendo al repositorio a través de las URLs `file:///`.

Servidores y Permisos: Una Palabra de Advertencia

Primero, recuerde que un repositorio Subversion es una colección de archivos de la base de datos BerkeleyDB; cualquier proceso que acceda al repositorio en forma directa necesita tener permisos apropiados de lectura y escritura sobre el repositorio entero. Si usted no es cuidadoso, puede causarle grandes dolores de cabeza. Siéntase seguro de haber leído “Ofrecer múltiples métodos de acceso al repositorio”.

En segundo lugar, cuando configure **svnserve**, Apache **httpd**, o cualquier otro proceso servidor, tenga en mente que podrá no querer ejecutar el proceso de servidor como el usuario `root` (o como cualquier otro usuario que tenga permisos ilimitados). Dependiendo con el propietario y los permisos del repositorio con los que lo haya exportado, con frecuencia es prudente utilizar un usuario diferente—quizás personalizado—. Por ejemplo, muchos administradores crean un usuario llamado `svn`, otorgándole la propiedad y derechos exclusivos sobre los repositorios de Subversion exportados, y únicamente se ejecutará los procesos de servidor con ese usuario.

Una vez el programa **svnserve** se esté ejecutando, hace que cada repositorio en su sistema se encuentre disponibles a la red. Un cliente necesita especificar una ruta *absoluta* en la URL del repositorio. Por ejemplo, si un repositorio se encuentra ubicado en `/usr/local/repositories/project1`, el cliente accederá a través de `svn://host.example.com/usr/local/repositories/project1` . Para incrementar la seguridad, usted puede pasar la opción `-r` a **svnserve**, que limita a exportar sólo los repositorios debajo de esa ruta:

```
$ svnserve -d -r /usr/local/repositories
…
```

En efecto, utilizando la opción `-r` modifica la ubicación que el programa tratará como raíz del espacio de sistema de archivos remoto. En consecuencia, los clientes utilizarán URLs que tengan la porción que le ha sido removida de la ruta, dejando URLs mucho más cortas (y mucho menos reveladoras):

```
$ svn checkout svn://host.example.com/project1
…
```

Autenticación y autorización integradas

Cuando un cliente se conecta a un proceso **svnserve**, las siguientes cosas suceden:

- El cliente selecciona un repositorio específico.
- El servidor procesa el archivo `conf/svnserve.conf` del repositorio, y comienza a poner en ejecución cualquier política de autenticación y autorización definida en éste.
- Dependiendo de la situación y las políticas de autorización,
  - se le puede permitir al cliente realizar pedidos de forma anónima, y sin haber recibido la demanda de autenticación, O
  - el cliente puede ser consultado por la autenticación en cualquier momento, O
  - si opera en el "modo túnel", el cliente se anuncia a sí mismo que ya ha sido autenticado externamente.

En el momento de escribir estas líneas, el servidor sólo sabe cómo realizar la autenticación por CRAM-MD5 [4]. En esencia, el servidor envía unos pocos datos al cliente. El cliente usa el algoritmo hash MD5 para crear una huella digital de los datos y la palabra clave combinados, y envía la huella digital como respuesta. El servidor realiza el mismo cálculo con la palabra clave almacenada para verificar que el resultado es idéntico. *En ningún momento el password real viaja a través de la red.*

También es posible, por supuesto, que el cliente se autentifique externamente vía agente de túnel , como por ejemplo **SSH**. En este caso, el servidor simplemente examina el usuario bajo el cual se está ejecutando, y lo usa durante la autenticación.

Tal y como habrá imaginado, el fichero `svnserve.conf` es el mecanismo central de un repositorio para controlar las políticas de autenticación y autorización. El fichero sigue el mismo formato que otros ficheros de configuración (vea "Área de configuración de parámetros de ejecución"): los nombres de sección están enmarcados por corchetes (`[` y `]`), los comentarios comienzan con almohadillas (#), y cada sección contiene variables específicas que pueden ser modificadas (`variable = valor`). Démonos un paseo por este fichero para aprender a usarlas.

Crear un fichero 'users' y realm

Por ahora, la sección `[general]` de `svnserve.conf` tiene todas las variables que usted necesita. Comencemos definiendo un fichero que contenga nombres de usuario y palabras clave, y una realm de autenticación:

```
[general]
password-db = userfile
realm = example realm
```

El nombre `realm` es algo que define usted. Le dice a los clientes a qué tipo de "espacio de nombres de autenticación" se están conectando; el cliente de Subversion lo muestra durante las preguntas de autenticación, y lo usa como palabra clave (junto con el nombre y puerto del servidor) para cachear las credenciales en disco (vea "Client Credentials Caching".) La variable `password-db` apunta a un fichero separado que contiene una lista de nombres de usuario y claves, usando el mismo formato familiar. Por ejemplo:

```
[users]
harry = foopassword
sally = barpassword
```

[4] Vea el RFC 2195.

El valor de `password-db` puede ser una ruta absoluta o relativa al fichero de usuarios. Para muchos administradores, es sencillo guardar el fichero justo en el área `conf/` del repositorio, junto a `svnserve.conf`. Por otra parte, es posible querer compartir el mismo fichero de usuarios entre dos o más repositorios; en este caso, el fichero probablemente debería ubicarse en un lugar más público. Los repositorios que compartan el fichero de usuarios también deberían estar configurados para usar la misma realm, dado que la lista de usuarios define esencialmente un realm de autenticación. Esté donde esté el fichero, asegúrese de ajustar apropiadamente los permisos de lectura y escritura. Si sabe bajo qué usuario(s) se ejecutará **svnserve**, restrinja el acceso de lectura al fichero correspondientemente.

Activar control de acceso

Hay dos variables más en el fichero `svnserve.conf`: determinan qué pueden hacer los usuarios autenticados y sin autenticar (anónimos). Las variables `anon-access` y `auth-access` pueden tener los valores `none`, `read`, o `write`. Poner el valor a `none` restringe el acceso de todo tipo; `read` permite únicamente el acceso de sólo lectura al repositorio, y `write` permite acceso completo de lectura y escritura. Por ejemplo:

```
[general]
password-db = userfile
realm = example realm

# anonymous users can only read the repository
anon-access = read

# authenticated users can both read and write
auth-access = write
```

Los parámetros del ejemplo, son de hecho los valores por defecto de las variables, en caso de que olvide definirlas. Si desea ser aun más conservativo, puede bloquear el acceso anónimo completamente:

```
[general]
password-db = userfile
realm = example realm

# anonymous users aren't allowed
anon-access = none

# authenticated users can both read and write
auth-access = write
```

Fíjese que **svnserve** sólo entiende de control de acceso “básico”. Un usuario tiene acceso universal de lectura escritura, acceso universal de lectura, o ningún tipo de acceso. No hay control detallado del acceso a rutas específicas dentro del repositorio. Para muchos proyectos y equipos, este nivel de control de acceso es más que suficiente. No obstante, si necesita control de acceso por directorio, tendrá que usar Apache en lugar de **svnserve** como su proceso servidor.

Autenticación y autorización SSH

La autenticación de serie de **svnserve** puede ser muy útil, porque evita tener que crear cuentas de sistema reales. Por otro lado, algunos administradores ya tienen entornos de autenticación SSH bien establecidos y funcionando. En estas situaciones, todos los usuarios del proyecto tienen cuentas en el sistema y la habilidad para conectarse con SSH a la máquina servidora.

Es fácil usar SSH junto con **svnserve**. El cliente simplemente usa el esquema de URL `svn+ssh://` para conectar:

```
$ whoami
harry
```

```
$ svn list svn+ssh://host.example.com/repos/project
harry@host.example.com's password:  *****

foo
bar
baz
…
```

Lo que sucede en esta situación es que el cliente de Subversion invoca el proceso local **ssh**, conectándolo con `host.example.com`, autenticándose como el usuario `juan`, y entonces lanzando un proceso **svnserve** privado en la máquina remota, que se ejecuta como el usuario `juan`. El comando **svnserve** se invoca en modo túnel (`-t`) y todo el protocolo de red es canalizado a través "tunneled" de la conexión cifrada por **ssh**, el agente del túnel. **svnserve** se da cuenta de que se está ejecutando como el usuario `juan`, y si el cliente realiza cambios en el repositorio, el nombre de usuario autenticado será usado para atribuir la autoría de la nueva revisión.

Cuando se usa un túnel, la autorización es principalmente controlada por los permisos del sistema operativo a los ficheros de la base de datos del repositorio; es casi igual que si Juan estuviese accediendo al repositorio directamente vía URL `file:///`. Si necesita que múltiples usuarios del sistema accedan al repositorio directamente, podría ponerlos en un mismo grupo, y ajustar cuidadosamente sus umasks. (Asegúrese de haber leído "Ofrecer múltiples métodos de acceso al repositorio".) Pero incluso en el caso de usar túneles, el fichero `svnserve.conf` todavía puede ser usado para bloquear el acceso, simplemente usando `auth-access = read` o `auth-access = none`.

Podría pensar que la historia del túnel por SSH acabaría aquí, pero no lo hace. Subversion le permite personalizar el comportamiento de los túneles en su fichero de configuración `config` (vea "Área de configuración de parámetros de ejecución".) Por ejemplo, supongamos que desea usar RSH en lugar de SSH. En la sección `[tunnels]` de su fichero `config`, defina lo siguiente:

```
[tunnels]
rsh = rsh
```

Y ahora, usted puede usar esta nueva definición de túnel usando un esquema URL que concuerde con el nombre de su nueva variable: `svn+rsh://host/path`. Cuando use el nuevo esquema URL, el cliente Subversion estará ejecutando realmente el comando **rsh host svnserve -t** tras las cortinas. Si incluye un nombre de usuario en la URL (por ejemplo, `svn+rsh://nombreusuario@host/path`) el cliente también lo incluirá en su comando (**rsh nombreusuario@host svnserve -t**.) Pero puede definir nuevos esquemas de túneles mucho más elaborados que eso:

```
[tunnels]
joessh = $JOESSH /opt/alternate/ssh -p 29934
```

Este ejemplo muestra varias cosas. Primero, enseña cómo hacer que el cliente de Subversion lance un binario de túnel muy específico (uno ubicado en `/opt/alternate/ssh`) con opciones específicas. En este caso, acceder a la URL `svn+joessh://` invocaría este particular binario de SSH con `-p 29934` como parámetro—útil si desea que el programa de túnel se conecte a un puerto no estándar.

Segundo, enseña cómo definir una nueva variable de entorno personalizada que puede reemplazar el nombre de un programa de túnel. Modificar la variable de entorno `SVN_SSH` es un modo conveniente de reemplazar el agente de túneles SSH por defecto. Pero si necesita tener diferentes reemplazos para servidores diferentes, cada uno de ellos conectando quizás a un puerto diferente o usando un conjunto diferente de opciones, puede usar el mecanismo demostrado en este ejemplo. Ahora, si fuese a crear la variable de entorno `JOESSH`, su valor reemplazaría el valor completo de la variable del túnel—se ejecutaría $JOESSH en lugar de `/opt/alternate/ssh -p 29934`.

httpd, el servidor HTTP Apache

El servidor HTTP apache es un servidor de red "heavy duty" que Subversion puede aprovechar. A través de un módulo propio,

httpd permite servir a clientes repositorios Subversion por el protocolo WebDAV/DeltaV, el cual es una extensión sobre HTTP 1.1 (vea `http://www.webdav.org/` para más información.) Este protocolo coge el ubicuo protocolo HTTP, núcleo de la World Wide Web, y añade la capacidad de escritura—específicamente el versionado de la misma. El resultado es un sistema robusto, estandarizado, empaquetado convenientemente como parte del software Apache 2.0, soportado por numerosos sistemas operativos y productos de terceros, y no necesita que sus administradores de red abran otro puerto adicional. [5] Si bien el servidor Apache-Subversion tiene más características que **svnserve**, también es más difícil de configurar. La flexibilidad a menudo se paga en complejidad.

Buena parte del texto siguiente incluye referencias a las directivas de configuración de Apache. Aunque proporcionamos algunos ejemplos que muestran el uso de estas directivas, describirlas con todo detalle está fuera del alcance de este capítulo. El equipo de Apache mantiene una documentación excelente, disponible públicamente en su página web en `http://httpd.apache.org`. Por ejemplo, una referencia general de las directivas de configuración está ubicada en `http://httpd.apache.org/docs-2.0/mod/directives.html`.

También, a medida que vaya realizando cambios en su configuración de Apache, es probable que en alguna parte cometerá algún fallo. Si no está familiarizado con el subsistema de archivos de registro de Apache, debería dedicarle un tiempo. En su fichero `httpd.conf` hay directivas que especifican ubicaciones físicas de los archivos de registro de acceso y error generados por apache (las directivas `CustomLog` y `ErrorLog` respectivamente). El módulo mod_dav_svn de Subversion usa también la interfaz de registro de mensajes de error de Apache. Siempre puede explorar el contenido de esos ficheros para encontrar información que podría revelar la fuente de un problema difícilmente discernible de otro modo.

¿Por qué Apache 2?

Si usted es un administrador de sistemas, es muy probable que ya tenga funcionando un servidor web Apache y cuente con algo de experiencia con él. En el momento de escribir estas líneas, Apache 1.3 es de largo la versión más popular. El mundo ha sido algo lento en la actualización a las series 2.X por varias razones: algunas personas temen los cambios, especialmente cambiar algo tan crítico como un servidor web. Otras personas demandan los módulos plug-in que sólo funcionan con la API de Apache 1.3, y están esperando que sean portados a la 2.X. Sea la razón que sea, muchas personas comienzan a preocuparse cuando descubren por primera vez que el módulo Subversion de Apache está escrito para la API 2 de Apache.

La respuesta correcta a este problema es: no se preocupe por ello. Es fácil ejecutar Apache 1.3 y Apache 2 simultáneamente; simplemente instálelos en lugares diferentes, y use Apache 2 como un servidor dedicado a Subversion que se ejecuta en un puerto diferente al 80. Los clientes podrán acceder al repositorio indicando el puerto en la URL:

```
$ svn checkout http://host.example.com:7382/repos/project
…
```

Requisitos previos

Para hacer disponible su repositorio por HTTP, básicamente necesita cuatro componentes disponibles en dos paquetes. Necesita el comando **httpd** de Apache 2.0, el módulo DAV **mod_dav** que viene con él, Subversion, y el módulo **mod_dav_svn**, que proporciona sistemas de ficheros, distribuido con Subversion. Una vez tenga todos estos componentes, el proceso para hacer disponible su repositorio es tan simple como:

- preparar y ejecutar httpd 2.0 con el módulo mod_dav,
- instalar el plug-in mod_dav_svn para mod_dav, el cual usa las librerías de Subversion para acceder a un repositorio, y
- configurar su fichero `httpd.conf` para exportar (o exponer) el repositorio.

[5] Realmente odian hacer eso.

Puede realizar los dos primeros pasos ya sea compilando **httpd** y Subversion a partir del código fuente, o instalando paquetes binarios precompilados en su sistema. Puede encontrar la información más actualizada sobre cómo compilar Subversion para su uso con el servidor HTTP Apache, al igual que cómo compilar y configurar Apache para este propósito, en el fichero `INSTALL` en el directorio raíz del código fuente de Subversion.

Configuración básica de Apache

Una vez tenga todos los componentes necesarios instalados en su sistema, todo lo que queda por hacer es configurar Apache mediante su fichero `httpd.conf`. Ordene a Apache que cargue el módulo mod_dav_svn module usando la directiva `LoadModule`. La directiva debe preceder cualquier otro elemento de configuración relacionado con Subversion. Si su Apache fue instalado usando el esquema por defecto, su módulo **mod_dav_svn** debería haber sido instalado en el subdirectorio `modules` de la instalación (a menudo `/usr/local/apache2`). La directiva `LoadModule` tiene una sintaxis sencilla, relacionando el nombre de un módulo con la ubicación de una librería dinámica en disco: disk:

```
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

Tenga en cuenta que si **mod_dav** fue compilado como un objeto compartido (en lugar de haber sido enlazado de manera estática con el binario **httpd**), necesitará también una línea `LoadModule` similar para él. Asegúrese de que aparece antes de la línea **mod_dav_svn**:

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

En un lugar posterior de su fichero de configuración, necesita indicarle a Apache dónde guardará su repositorio (o repositorios) de Subversion. La directiva `Location` sigue una notación tipo XML, comenzando con una etiqueta de apertura y terminando con una de cierre, con varias otras directivas de configuración en medio. El propósito de la directiva `Location` es indicar a Apache que debe realizar algo especial cuando tenga que procesar peticiones dirigidas a una URL determinada o a una hija suya. En el caso de Subversion, quiere que Apache simplemente le pase el control a la capa DAV de todas las URLs que apunten a recursos versionados. Puede ordenar a Apache que delegue el control de todas las URLs cuyas porciones de rutas (aquella parte de la URL que sigue el nombre del servidor y número de puerto opcional) empiecen con `/repos/` a un proveedor DAV cuyo repositorio se encuentre en `/ruta/absoluta/al/repositorio` usando la siguiente sintaxis de `httpd.conf`:

```
<Location /repos>
  DAV svn
  SVNPath /ruta/absoluta/al/repositorio
</Location>
```

Si planea soportar múltiples repositorios de Subversion que residirán en el mismo directorio padre de su disco local, puede usar una directiva alternativa, la directiva `SVNParentPath`, para indicar ese directorio padre común. Por ejemplo, si sabe que creará múltiples repositorios Subversion en el directorio `/usr/local/svn` que sería accesible mediante URLs como `http://my.server.com/svn/repos1`, `http://my.server.com/svn/repos2`, y así en adelante, podría usar la sintaxis de configuración `httpd.conf` del siguiente ejemplo:

```
<Location /svn>
  DAV svn

  # any "/svn/foo" URL will map to a repository /usr/local/svn/foo
  SVNParentPath /usr/local/svn
</Location>
```

Usando la sintaxis anterior, Apache delegará el control de todas las URLs cuyas porciones comiencen con `/svn/` al proveedor DAV de Subversion, quien asumirá que todos los elementos en el directorio especificado por la directiva `SVNParentPath` son

en realidad repositorios de Subversion. Esta es una sintaxis particularmente conveniente, ya que a diferencia de la directiva `SVNPath`, no necesita reiniciar Apache para crear y hacer disponibles nuevos repositorios.

Asegúrese de que cuando defina su nuevo `Location`, no se solapa con otras ubicaciones exportadas. Por ejemplo, si su `DocumentRoot` principal es `/www`, no exporte ningún repositorio Subversion en `<Location /www/repos>`. En caso de recibir una petición para la URI `/www/repos/foo.c`, Apache no sabrá si tiene que buscar el fichero `repos/foo.c` en `DocumentRoot`, o si debe delegar en **mod_dav_svn** para que devuelva `foo.c` del repositorio Subversion.

Nombres de servidor y peticiones COPY

Subversion hace uso del tipo de petición `COPY` para realizar copias de ficheros y directorios en el lado del servidor. Como parte de los chequeos de seguridad realizados por los módulos Apache, se espera que el origen de la copia esté ubicado en la misma máquina que el destino. Para satisfacer este requisito, podría necesitar decirle a mod_dav el nombre de servidor que usa para su máquina. En general, puede usar la directiva `ServerName` de `httpd.conf` para lograr esto.

```
ServerName svn.example.com
```

Si está usando el soporte de hosting virtual de Apache por medio de la directiva `NameVirtualHost`, puede que necesite usar la directiva `ServerAlias` para especificar nombres adicionales bajo los cuales es conocido su servidor. De nuevo, busque en la documentación de Apache los detalles de la misma.

En este punto, debería considerar seriamente el tema de los permisos. Si ha estado ejecutando Apache durante algún tiempo como su servidor de web habitual, probablemente ya tenga una colección de contenido—páginas web, scripts y demás. Estos elementos ya han sido configurados con un conjunto de permisos que les permite funcionar con Apache, o de manera más apropiada, le permite a Apache funcionar con esos ficheros. Apache, cuando es usado como servidor de Subversion, también necesitará los permisos adecuados de lectura y escritura a su repositorio Subversion. (Vea Servidores y Permisos: Una Palabra de Advertencia.)

Tendrá que encontrar una configuración de permisos de sistema que satisfaga los requisitos de Subversion sin estropear instalaciones ya existentes de páginas web o scripts. Esto podría significar cambiar los permisos de su repositorio Subversion para que encajen con aquellos usados por otras elementos servidos por Apache, o podría significar usar las directivas `User` y `Group` del fichero `httpd.conf` para especificar que Apache debe ejecutarse con el usuario y grupo a quien pertenece su repositorio Subversion. No hay una única manera correcta de configurar sus permisos, y cada administrador tendrá diferentes razones para hacer las cosas a su manera. Sólo tenga presente que los problemas relacionados con los permisos son quizás el fallo más común cuando se configura un repositorio Subversion para ser usado con Apache.

Opciones de autenticación

En este punto, si ha configurado `httpd.conf` para que contenga algo como

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
</Location>
```

...entonces su repositorio es accesible al mundo de manera “anónima”. Hasta que configure algunas políticas de autenticación y autorización, los repositorios Subversion que haga disponibles con la directiva `Location` serán generalmente accesibles por cualquiera. En otras palabras,

- cualquiera puede usar su cliente de Subversion para obtener una copia local de la URL de un repositorio (o cualquiera de sus subdirectorios),
-

cualquiera puede navegar de manera interactiva las últimas revisiones del repositorio simplemente usando la URL del repositorio con su navegador web, y

- cualquiera puede enviar cambios al repositorio.

Autenticación HTTP básica

La forma más sencilla de autenticar a un cliente es mediante el mecanismo de autenticación HTTP básico, que simplemente usa un nombre de usuario y clave para verificar que el usuario es quien dice ser. Apache proporciona la utilidad **htpasswd** para gestionar la lista de nombres de usuarios y claves aceptables, aquellos a quienes desea proporcionar acceso especial a su repositorio Subversion. Permitamos el acceso de escritura a Juan y Carmen. Primero, necesitamos añadirlos al fichero de claves.

```
$ ### First time: use -c to create the file
$ ### Use -m to use MD5 encryption of the password, which is more secure
$ htpasswd -cm /etc/svn-auth-file harry
New password: *****
Re-type new password: *****
Adding password for user harry
$ htpasswd /etc/svn-auth-file -m sally
New password: *******
Re-type new password: *******
Adding password for user sally
$
```

Ahora, necesitamos añadir algunas directivas `httpd.conf` más dentro del bloque `Location` para decirle a Apache qué hacer con su nuevo fichero de claves. La directiva `AuthType` especifica el tipo del sistema de autenticación que va a usar. En este caso, queremos especificar el sistema de autenticación `Basic`. `AuthName` es un nombre arbitrario que usted proporciona para el dominio de autenticación. La mayoría de los navegadores mostrarán este nombre en una ventana de diálogo emergente cuando el navegador le pregunte su nombre y clave. Finalmente, use la directiva `AuthUserFile` para especificar la ubicación del fichero de claves creado usando **htpasswd**.

Tras añadir estas tres directivas, su bloque `<Location>` debería tener un aspecto similar a este:

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /etc/svn-auth-file
</Location>
```

Este bloque `<Location>` todavía no está completo, y no hará nada útil. Meramente indica a Apache que cuando requiera autorización, Apache debe obtener el nombre de usuario y clave del cliente Subversion. Lo que falta aquí, no obstante, son las directivas que le dicen a Apache *qué* tipos de solicitudes por parte de los clientes requieren autorización. Donde se requiera autorización, Apache demandará también autenticación. Lo mas sencillo es proteger todas las peticiones. Añadir `Require valid-user` indica a Apache que todas las peticiones requieren un usuario autenticado:

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /etc/svn-auth-file
```

```
  Require valid-user
</Location>
```

Asegúrese de leer la siguiente sección (“Opciones de autorización”) para más detalles sobre la directiva `Require` y otros modos de establecer políticas de autorización.

Una advertencia: en la autenticación básica por HTTP las palabras claves viajan por la red casi como texto simple, y son por lo tanto extremadamente inseguras. Si está preocupado por la existencia de un rastreador de claves, lo mejor sería usar algún tipo de cifrado SSL, para que los clientes se autentiquen vía `https://` en lugar de `http://`; como mínimo podrá configurar Apache para que use use un certificado firmado por sí mismo. [6] Consulte la documentación de Apache (y la de OpenSSL) para saber cómo hacer esto.

Gestión de certificados SSL

Las empresas que necesitan exponer sus repositorios a un acceso externo al firewall de la compañía deberían ser conscientes de que grupos no autorizados podrían estar “rastreando” su tráfico de red. SSL consigue que ese tipo de atención no deseada desemboque con menor probabilidad en fugas de datos sensitivos.

Si se compila un cliente de Subversion para que use OpenSSL, entonces ganará la habilidad de hablar con un servidor Apache vía URLs `https://`. La librería Neon usada por el cliente Subversion no solo es capaz de verificar los certificados de servidor, sino que también puede proporcionar certificados de cliente cuando sean solicitados. Una vez el cliente y servidor hayan intercambiado sus certificados SSL y realizado una autenticación mutua con éxito, todas sus comunicaciones siguientes serán cifradas con una clave de sesión.

Está fuera del alcance de este libro describir la generación de certificados de cliente y servidor, y cómo configurar Apache para usarlos. Muchos otros libros, incluyendo la propia documentación de Apache, describen esta tarea. Pero lo que *podemos* cubrir aquí es cómo gestionar los certificados de cliente y servidor desde un cliente Subversion ordinario.

Cuando un cliente Subversion se comunica con Apache vía `https://`, puede recibir dos tipos diferentes de información:

- un certificado de servidor
- una demanda de un certificado de cliente

Si el cliente recibe un certificado de servidor, necesita verificar que confía en el certificado: ¿es realmente el servidor quien dice ser? La librería OpenSSL realiza esto examinando al firmante del certificado de servidor o la *autoridad certificadora* (CA). Si OpenSSL no es capaz de confiar automáticamente en la CA, o bien ocurre algún otro problema (como que el certificado expiró o no concuerda con el nombre del servidor), el cliente de línea de comando de Subversion le preguntará si quiere confiar en el certificado del servidor de todas maneras:

```
$ svn list https://host.example.com/repos/project

Error validating server certificate for 'https://home.example.com:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
 - Hostname: host.example.com
 - Valid: from Jan 30 19:23:56 2004 GMT until Jan 30 19:23:56 2006 GMT
 - Issuer: CA, example.com, Sometown, California, US
 - Fingerprint: 7d:e1:a9:34:33:39:ba:6a:e9:a5:c4:22:98:7b:76:5c:92:a0:9c:7b
```

[6] Aunque los certificados de servidor auto firmados siguen siendo vulnerables a ataques del “hombre en medio” , tal ataque es todavía más difícil de conseguir por un observador casual, comparado con el rastreado de claves desprotegidas.

```
(R)eject, accept (t)emporarily or accept (p)ermanently?
```

Este diálogo debería parecerle familiar; es esencialmente la misma pregunta que ha observado proveniente de su navegador web (¡que es meramente otro cliente HTTP como Subversion!). Si elige la opción (p)ermanente, el certificado de servidor será guardado en el área `auth/` de su fichero privado de configuración de parámetros de ejecución, del mismo modo que es guardado su nombre de usuario y clave (vea “Client Credentials Caching”.) Si se cachea, Subversion recordará confiar de manera automática en este certificado en futuras negociaciones.

Su fichero `servers` del área de la configuración de parámetros de ejecución también le permite a su cliente de Subversion confiar de manera automática en CAs específicos, ya sea de manera global o individual. Simplemente modifique la variable `ssl-authority-files` para que sea una lista de certificados de CA con codificación PEM separados por punto y coma:

```
[global]
ssl-authority-files = /path/to/CAcert1.pem;/path/to/CAcert2.pem
```

Muchas instalaciones de OpenSSL confían “por defecto” de manera casi universal en un conjunto predefinido de CAs. para hacer que el cliente Subversion confíe de manera automática en estas autoridades estándar, cambie la variable `ssl-trust-default-ca` a `true`.

Cuando dialoga con Apache, un cliente de Subversion también podría recibir una petición de certificado de cliente. Apache está solicitando al cliente que se identifique: ¿es el cliente realmente quien dice ser? Si todo va bien, el cliente de Subversion devuelve un certificado privado firmado por la CA en quien confía Apache. El certificado de cliente habitualmente se almacena cifrado en el disco, protegido por una palabra clave local. Cuando Subversion recibe esta petición, le preguntará tanto por la ruta al certificado como por la palabra clave que lo protege:

```
$ svn list https://host.example.com/repos/project

Authentication realm: https://host.example.com:443
Client certificate filename: /path/to/my/cert.p12
Passphrase for '/path/to/my/cert.p12':  ********
…
```

Tenga que el certificado de un cliente es un fichero “p12”. Para usar un certificado de cliente con Subversion, debe estar en el formato PKCS#12, que es un estándar portable. La mayoría de los navegadores son capaces de importar y exportar certificados en este formato. Otra opción es usar las herramientas de línea de comando de OpenSSL para convertir certificados existentes a PKCS#12.

De nuevo, el fichero `servers` del área de configuración de parámetros de ejecución le permite automatizar este proceso por cada máquina. Una o ambas informaciones pueden ser descritas en variables de ejecución:

```
[groups]
examplehost = host.example.com

[examplehost]
ssl-client-cert-file = /path/to/my/cert.p12
ssl-client-cert-password = somepassword
```

Una vez creadas las variables `ssl-client-cert-file` y `ssl-client-cert-password`, el cliente Subversion responderá automáticamente a peticiones de certificado de cliente sin tener que preguntarle. [7]

[7] Aquellos más conscientes de la seguridad quizás no deseen almacenar la clave del certificado de cliente en el fichero `servers` del área de configuración de parámetros de ejecución.

Opciones de autorización

En este punto, ya ha configurado la autenticación, pero no la autorización. Apache es capaz de cuestionar a los clientes y confirmar sus identidades, pero todavía no sabe cómo permitir o restringir el acceso a los clientes con esas identidades. Esta sección describe dos estrategias para controlar el acceso a sus repositorios.

Control de acceso simple

La forma de control de acceso más simple a un repositorio es autorizar a ciertos usuarios el acceso de sólo lectura, o lectura y escritura.

Puede restringir el acceso sobre todas las operaciones del repositorio añadiendo la directiva `Require valid-user` al bloque `<Location>`. Usando nuestro ejemplo anterior, esto equivaldría a que sólo los clientes que clamasen ser `juan` o `carmen`, y proporcionan la clave correcta para su usuario correspondiente, podrían hacer operaciones con el repositorio Subversion:

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file

  # only authenticated users may access the repository
  Require valid-user
</Location>
```

A veces no necesita extremar tanto la seguridad. Por ejemplo, el propio repositorio de código fuente de Subversion ubicado en `http://svn.collab.net/repos/svn` permite a cualquiera realizar operaciones de sólo lectura (como por ejemplo obtener copias locales y explorar el repositorio con un navegador web), pero restringe todas las operaciones de escritura a usuarios autenticados. Para realizar este tipo de selección restrictiva, puede usar las directivas de configuración `Limit` y `LimitExcept`. Igual que la directiva `Location`, estos bloques tienen etiquetas de comienzo y final, y las anidaría dentro de su bloque `<Location>`.

Los parámetros presentes en las directivas `Limit` y `LimitExcept` son los tipos de peticiones HTTP afectadas por ese bloque. Por ejemplo, si quisiese prohibir todo el acceso a su repositorio excepto las operaciones actualmente soportadas de sólo lectura, podría usar la directiva `LimitExcept` incluyendo los tipos de peticiones `GET`, `PROPFIND`, `OPTIONS`, y `REPORT`. Entonces la directiva anteriormente mencionada `Require valid-user` sería colocada dentro del bloque `<LimitExcept>` en lugar del bloque.

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file

  # For any operations other than these, require an authenticated user.
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>
</Location>
```

Estos son sólo unos pocos ejemplos simples. Para más información detallada sobre el control de acceso de Apache y la directiva `Require`, échele un vistazo a la sección `Security` de la colección de tutoriales en la sección de documentación de Apache en

`http://httpd.apache.org/docs-2.0/misc/tutorials.html`.

Control de acceso por directorio

Es posible establecer permisos más precisos usando el segundo módulo de Apache httpd, **mod_authz_svn**. Este módulo coge las diversas URLs opacas del cliente al servidor, solicita a **mod_dav_svn** que las decodifique, y entonces posiblemente veta estas peticiones basándose en la política definida en un fichero de configuración.

Si ha instalado Subversion a partir del código fuente, **mod_authz_svn** habrá sido instalado automáticamente junto a **mod_dav_svn**. Muchas distribuciones binarias también lo instalan automáticamente. Para verificar que está instalado correctamente, asegúrese de que viene justo tras la directiva `LoadModule` de **mod_dav_svn** en `httpd.conf`:

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
LoadModule authz_svn_module   modules/mod_authz_svn.so
```

Para activar este módulo, necesita configurar su bloque `Location` para que use la directiva `AuthzSVNAccessFile`, la cual especifica el fichero que contiene la política de permisos para las rutas en sus repositorios. (Discutiremos el formato de este fichero dentro de unos momentos.)

Apache es flexible, así que tiene la opción de configurar su bloque con uno de estos tres patrones generales. Para comenzar, escoja uno de estos patrones básicos de configuración. (Los ejemplos que vienen a continuación son muy simples; vea la documentación de Apache para más detalles sobre las opciones de autenticación y autorización.)

El bloque más simple es permitir el acceso a cualquiera. En este escenario, Apache nunca envía solicitudes de autenticación, así que todos los usuarios son tratados como “anónimos”.

Ejemplo 6.1. A sample configuration for anonymous access.

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file
</Location>
```

En el lado opuesto de la escala de paranoia, puede configurar su bloque para demandar la autenticación de todo el mundo. Todos los clientes deberán proporcionar credenciales para identificarse. Su bloque requiere de manera incondicional la autenticación vía directiva `Require valid-user`, y define un método de autenticación.

Ejemplo 6.2. A sample configuration for authenticated access.

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file

  # only authenticated users may access the repository
```

```
  Require valid-user

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file
</Location>
```

Un tercer patrón muy popular es permitir una combinación de acceso autenticado y anónimo. Por ejemplo, muchos administradores quieren que los usuarios anónimos puedan leer ciertos directorios del repositorio, pero restringen la lectura (o escritura) de otras áreas más sensibles a usuarios autenticados. Con esta configuración, todos los usuarios comienzan accediendo al repositorio de manera anónima. Si su política de control de acceso requiere un nombre de usuario real en algún punto, Apache demandará una autenticación del cliente. Para hacer esto, use tanto la directiva `Satisfy Any` como `Require valid-user` simultáneamente.

Ejemplo 6.3. A sample configuration for mixed authenticated/anonymous access.

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/access/file

  # try anonymous access first, resort to real
  # authentication if necessary.
  Satisfy Any
  Require valid-user

  # how to authenticate a user
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /path/to/users/file
</Location>
```

Una vez tenga configurado su bloque `Location` básico, puede crear un fichero de acceso y definir en él algunas reglas de autorización.

La sintaxis del fichero de acceso es la misma que la usada por **svnserve.conf** y los ficheros del área de configuración de parámetros de ejecución. Las líneas que comienzan con una almohadilla (#) son ignoradas. En su forma más simple, cada sección nombra a un repositorio y ruta dentro del mismo, y los nombres de usuario autenticados son los nombres de las opciones dentro de cada sección. El valor de cada opción describe el nivel de acceso del usuario a la ruta del repositorio: ya sea `r` (sólo lectura) o `rw` (lectura-escritura). Si el nombre de un usuario no se menciona, no se le permite el acceso.

Para ser más específicos, el valor de los nombres de sección tienen la forma `[repositorio-nombre:ruta]` o la forma `[ruta]`. Si está usando la directiva `SVNParentPath`, entonces es importante especificar los nombres de repositorio en sus secciones. Si los omite, una sección como `[/algún/directorio]` coincidirá con la ruta `/algún/directorio` en *cada* repositorio. No obstante si está usando la directiva `SVNPath`, entonces está bien definir rutas en sus secciones—después de todo, sólo hay un repositorio.

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
```

En este primer ejemplo, el usuario `juan` tiene permisos de lectura y escritura sobre el directorio `/branches/calc/bug-142` del repositorio `calc`, pero el usuario `carmen` sólo tiene permisos de lectura. Cualquier otro usuario tendrá bloqueado el acceso a este directorio.

Por supuesto, los permisos son heredados del directorio padre al hijo. Esto significa que puede especificar un subdirectorio con una política de acceso diferente para Carmen:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

# give sally write access only to the 'testing' subdir
[calc:/branches/calc/bug-142/testing]
sally = rw
```

Ahora Carmen puede escribir en el subdirectorio `testing` de la rama, pero sigue accediendo con permisos de sólo lectura a otras ubicaciones. Mientras tanto, Juan continua con permisos de lectura-escritura para toda la rama.

También es posible denegarle los permisos a alguien de manera explícita mediante reglas de herencia, ajustando su variable de nombre de usuario a nada:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r

[calc:/branches/calc/bug-142/secret]
harry =
```

En este ejemplo, Juan tiene acceso de lectura-escritura a todo el árbol `bug-142`, pero no tiene acceso en absoluto al subdirectorio `secret` dentro del mismo.

Por defecto, nadie tiene acceso en absoluto al repositorio. Esto significa que si comienza con un fichero vacío, probablemente querrá permitir el acceso de lectura la raíz del repositorio a todos los usuarios. Puede hacer esto con la variable asterisco (`*`), la cual significa “todos los usuarios”:

```
[/]
* = r
```

Ésta es una configuración común; fíjese que no hay nombre de repositorio nombrado en el nombre de la sección. Esto hace todos los repositorios legibles por todos los usuarios, usando tanto `SVNPath` como `SVNParentPath`. Una vez todos los usuarios tienen acceso de lectura a los repositorios, puede dar permisos explícitos `rw` a determinados usuarios en subdirectorios específicos dentro de repositorios específicos.

La variable asterisco (`*`) también merece aquí una mención especial: es el *único* patrón que coincide con el usuario anónimo. Si ha configurado su bloque `Location` para permitir una mezcla de acceso anónimo y autenticado, todos los usuarios comienzan accediendo a Apache de forma anónima. **mod_authz_svn** busca el valor `*` definido para la ruta que se está accediendo; si no puede encontrar uno, entonces Apache demanda una autenticación real del cliente.

El fichero de acceso también le permite definir grupos enteros de usuarios, de manera muy similar al fichero `/etc/group` Unix:

```
[groups]
calc-developers = harry, sally, joe
```

```
paint-developers = frank, sally, jane
everyone = harry, sally, joe, frank, sally, jane
```

A los grupos se les puede facilitar el acceso igual que a los usuarios. Distíngalos con el prefijo “arroba” (@):

```
[calc:/projects/calc]
@calc-developers = rw

[paint:/projects/paint]
@paint-developers = rw
jane = r
```

...y eso es prácticamente todo lo que hay que hacer.

Regalitos extra

Hemos cubierto la mayoría de las opciones de autenticación y autorización para Apache y mod_dav_svn. Pero Apache proporciona algunas otras características interesantes.

Navegar por el repositorio

Uno de los beneficios más útiles de una configuración Apache/WebDAV de su repositorio Subversion es que las revisiones más recientes de sus ficheros y directorios versionados están disponibles inmediatamente para ser visualizados con un navegador web normal. Dado que Subversion usa URLs para identificar recursos versionados, aquellas URLs basadas en HTTP para acceder al repositorio pueden teclearse directamente en un navegador. Su navegador realizará una petición `GET` para esta URL, y dependiendo de que ésta represente un directorio o fichero versionado, mod_dav_svn responderá con un listado de directorio o con el contenido del fichero.

Dado que las URLs no contienen información sobre qué versión del recurso desea ver, mod_dav_svn responderá siempre con la versión más reciente. Esta funcionalidad tiene el maravilloso efecto secundario de que puede pasar a sus colegas referencias a documentos, y esas URLs siempre apuntarán a la última manifestación de los mismos. Por supuesto, también puede usar las URLs como hiperenlaces desde otras páginas web.

Generalmente obtendrá más utilidad del uso de URLs a ficheros versionados—después de todo, ahí es donde suele estar el contenido interesante. Pero quizás disponga de la ocasión de navegar por un listado de directorio de Subversion, y percibir que el HTML generado para mostrar el listado es muy básico, y desde luego no está pensado para ser estéticamente agradable (o incluso interesante). Para permitir la personalización de estas muestras de directorio, Subversion proporciona una característica de indexado XML. Una única directiva `SVNIndexXSLT` en el bloque `Location` de `httpd.conf` indicará a mod_dav_svn que genere la salida del listado de directorio en un XML, usando como referencia una hoja de estilo XSLT de su elección:

```
<Location /svn>
  DAV svn
  SVNParentPath /usr/local/svn
  SVNIndexXSLT "/svnindex.xsl"
  …
</Location>
```

Usando la directiva `SVNIndexXSLT` y una hoja de estilo XSLT creativa, puede hacer que sus listados de directorio concuerden con los esquemas de color e imágenes usados en otras partes de su página web. O, si lo prefiere, puede usar las hojas de estilo de ejemplo que proporciona la distribución de código fuente de Subversion en el directorio `tools/xslt/`. Tenga en cuenta que la ruta que apunta al directorio `SVNIndexXSLT` es en realidad una URL—¡los navegadores deben poder leer sus hojas de estilo para poder usarlas!

¿Puedo ver revisiones antiguas?

¿Con un navegador web ordinario? En una palabra: no. Al menos, no con **mod_dav_svn** como su única herramienta.

Su navegador web sólo sabe hablar HTTP ordinario. Eso significa que sólo sabe cómo hacer GET de URLs públicas, las cuales representan las últimas versiones de ficheros y directorios. De acuerdo con la especificación WebDAV/DeltaV, cada servidor define una sintaxis privada de URLs para versiones antiguas de los recursos, y ésta es opaca a los clientes. Para encontrar una versión anterior de un fichero, un cliente debe seguir un procedimiento específico para "descubrir" la URL correcta; el proceso conlleva realizar una serie de peticiones WebDAV PROPFIND y entender los conceptos DeltaV. Esto es algo que su navegador web simplemente no es capaz de hacer.

Así que para responder a su pregunta, una manera obvia para ver revisiones antiguas de ficheros y directorios es pasando el parámetro `--revision` a los comandos **svn list** y **svn cat**. No obstante, para ver revisiones antiguas con su navegador, puede usar software de terceros. Un buen ejemplo es ViewCVS (`http://viewcvs.sourceforge.net/`). Originalmente ViewCVS fue escrito para mostrar repositorios CVS por la web, y las últimas versiones de desarrollo (en el momento de escribir estas palabras) son capaces de entender también repositorios de Subversion.

Otras características

Algunas de las características proporcionadas por Apache en su rol de servidor web robusto también pueden ser aprovechadas para incrementar la funcionalidad o seguridad en Subversion. Subversion se comunica con Apache usando Neon, que es una librería HTTP/WebDAV genérica con soporte para mecanismos tales como SSL (Secure Socket Layer, discutidos anteriormente) y compresión Deflate (el mismo algoritmo usado por los programas **gzip** y **PKZIP** para "reducir" ficheros en bloques de datos de tamaño menor). Sólo necesita compilar el soporte de las características que desea tener en Subversion y Apache, y configurar adecuadamente estos programas para que las usen.

La compresión Deflate carga ligeramente al cliente y al servidor para comprimir y descomprimir transmisiones por red para minimizar el tamaño de los datos. En aquellos casos donde el ancho de banda de red es escaso, este tipo de compresión puede incrementar sustancialmente la velocidad con la que se comunican el servidor y el cliente. En casos extremos, la minimización de la transmisión por red puede marcar la diferencia entre una operación cuyo tiempo de espera es agotado o termina con éxito.

Menos interesantes, pero igualmente útiles, son otras características de la relación entre Apache y Subversion, como la capacidad de especificar un puerto específico (en lugar del puerto 80 por defecto de HTTP) o un nombre de dominio virtual mediante el cual se pueda acceder a un repositorio de Subversion, o la capacidad de acceder al repositorio a través de un proxy. Estas cosas las soporta Neon, así que Subversion las obtiene de manera gratuita.

Por último, dado que **mod_dav_svn** habla un dialecto semi completo de WebDAV/DeltaV, es posible acceder al repositorio mediante clientes DAV de terceros. La mayoría de los sistemas operativos modernos (Win32, OS X, y Linux) tienen soporte para montar un servidor DAV como un "recurso" estándar de red. Ésto es un tema complicado; para más detalles, lea Apéndice C, *WebDAV y autoversionado*.

Ofrecer múltiples métodos de acceso al repositorio

Le hemos enseñado a acceder a un repositorio de muchas maneras diferentes. Pero, ¿es posible—o seguro—acceder a su repositorio mediante múltiples métodos simultáneamente? La respuesta es afirmativa, siempre y cuando planifique un poco las cosas.

En cualquier instante, estos procesos pueden requerir acceso de lectura y escritura a su repositorio:

- usuarios normales del sistema (que se identifican como ellos mismos) usando un cliente de Subversion para acceder a los repositorios directamente vía URLs `file:///`;
- usuarios normales del sistema (que se identifican como ellos mismos) conectándose a procesos **svnserve** privados mediante se-

siones SSH, los cuales acceden al repositorio;

- un proceso **svnserve**—ya sea un demonio o uno invocado por **inetd**—ejecutándose como un usuario particular invariable;
- un proceso **httpd** de Apache, ejecutándose como un usuario particular invariable.

El problema más común que se encuentran los administradores es con la propiedad y permisos del repositorio. ¿Tiene cada proceso (o usuario) en la lista anterior permisos para leer y escribir los ficheros de la base de datos Berkeley? Asumiendo que tiene un sistema operativo tipo Unix, una opción simple sería colocar a cada usuario potencial del repositorio en un nuevo grupo `svn`, y hacer que el repositorio al completo esté poseído por este grupo. Pero ni si quiera eso es suficiente, porque un proceso puede escribir los ficheros de la base de datos usando una máscara de permisos hostil—aquella que impida el acceso a otros usuarios.

Así que el siguiente paso tras el establecimiento de un grupo común para los usuarios del repositorio es forzar a todos los procesos que acceden al repositorio a usar una máscara de permisos aceptable. Para los usuarios que acceden al repositorio directamente, puede envolver el programa **svn** en un script cuyo primer paso sea ejecutar **umask 002** y después ejecute el programa cliente **svn** auténtico. Puede escribir un script de envoltorio similar para el programa **svnserve**, y añadir el comando **umask 002** al propio fichero de arranque de Apache, `apachectl`. Por ejemplo:

```
$ cat /usr/local/bin/svn

#!/bin/sh

umask 002
/usr/local/subversion/bin/svn "$@"
```

Hay otro problema común en sistemas tipo Unix. A medida que se usa un repositorio, la base de datos Berkeley ocasionalmente crea nuevos ficheros de registro para sus transacciones. Incluso si el repositorio completo pertenece al grupo **svn**, estos nuevos ficheros no pertenecerán necesariamente al mismo grupo, lo cual crea de nuevo problemas de permisos para sus usuarios. Un buen método para corregir esto es activar el bit de grupo SUID en el directorio `db` del repositorio. Esto causa que todos los ficheros de registros creados tengan el mismo grupo que el directorio padre..

Una vez haya saltado por estos aros, su repositorio debería ser accesible por todos los procesos necesarios. Puede que parezca un poco lioso y complicado, pero los problemas de tener múltiples usuarios compartiendo acceso de escritura a ficheros comunes son clásicos, pocas veces solucionados de manera elegante.

Afortunadamente, la mayoría de los administradores de repositorios no tendrán nunca la *necesidad* de tener configuraciones tan complejas. Los usuarios que desean acceder a repositorios que residen en la misma máquina no están limitados a usar el las URLs de acceso `file://`—típicamente pueden contactar con el servidor HTTP de Apache o **svnserve** usando el nombre de servidor `localhost` en sus URLs `http://` o `svn://`. Y mantener múltiples procesos servidores para sus repositorios de Subversion es probablemente un dolo de cabeza más que algo necesario. ¡Le recomendamos que escoja el servidor que se ajuste mejor a sus necesidades y que se limite a él!

Lista de tareas para servidor svn+ssh://

Puede ser complicado hacer que un grupo de usuarios con cuentas SSH existentes compartan un repositorio sin problemas de permisos. Si se siente confuso respecto a todas las cosas que usted (como administrador) necesita realizar en un sistema tipo Unix, aquí tiene una lista de tareas que resumen algunas de las cosas discutidas en esta sección:

- Todos sus usuarios SSH necesitan ser capaces de leer y escribir en el repositorio. Ponga a todos los usuarios SSH en un único grupo. Haga que el propietario del repositorio sea ese grupo, y active los permisos de lectura/escritura para el mismo.

- Sus usuarios deben usar una máscara de permisos aceptable al acceder al repositorio. Asegúrese de que **svnserve** (`/usr/local/bin/svnserve`, o donde quiera que esté en `$PATH`) es en realidad un script envoltorio que ejecuta **umask 002** y luego el binario **svnserve** auténtico. Tome medidas similares con **svnlook** y **svnadmin**. Es decir, ejecútelos con una máscara de permisos aceptable, o envuélvalos tal y como se describe anteriormente.
- Cuando la base de datos Berkeley crea nuevos ficheros de registros, también necesitan pertenecer al mismo grupo, así que asegúrese de que ejecuta **chmod g+s** en el directorio `db` del repositorio.

Capítulo 7. Tópicos avanzados

Si ha estado leyendo este libro capítulo por capítulo, de principio a fin, debería haber adquirido suficiente conocimiento para usar el cliente de Subversion para realizar la mayoría de las operaciones comunes de control de versiones. Entiende cómo obtener una copia local de un repositorio Subversion. Se siente cómodo enviando y recibiendo cambios usando las funciones **svn commit** y **svn update**. Probablemente hasta haya desarrollado un reflejo que le hace ejecutar el comando **svn status** de forma casi inconsciente. A efectos prácticos, está preparado para usar Subversion en un entorno típico.

Pero el conjunto de características de Subversion no acaba en "operaciones comunes de control de versiones".

Este capítulo recalca algunas características de Subversion que no son usadas muy a menudo. En él, discutiremos el soporte de propiedades (o "meta datos") de Subversion, y cómo modificar el comportamiento por defecto de Subversion alterando su área de configuración dedicada a parámetros de ejecución. Describiremos cómo puede usar definiciones externas para ordenar a Subversion que obtenga datos de múltiples repositorios. Cubriremos con detalle algunas herramientas adicionales, tanto del lado del cliente como del servidor, que son parte de la distribución de Subversion.

Antes de leer este capítulo, debe estar familiarizado con las capacidades básicas de versionado de ficheros y directorios de Subversion. Si todavía no ha leído sobre éstas, o si necesita refrescar su memoria, recomendamos que vea Capítulo 2, *Conceptos básicos* y Capítulo 3, *Recorrido guiado*. Una vez haya dominado lo básico y consumido este capítulo, ¡será un usuario avanzado de Subversion!

Área de configuración de parámetros de ejecución

Subversion proporciona muchos comportamientos opcionales que pueden ser controlados por el usuario. Muchas de estas opciones son del tipo que el usuario querría aplicar a todas las operaciones de Subversion. Así que en lugar de obligar a los usuarios a recordar parámetros de línea de comando para especificar estas opciones, y usarlos para todas y cada una de las operaciones que realicen, Subversion usa ficheros de configuración, segregados en un área de configuración de Subversion.

El *área de configuración* de Subversion es una jerarquía de dos capas de nombres de opciones y sus respectivos valores. Normalmente, esto se reduce a un directorio especial que contiene *ficheros de configuración* (la primera capa), que son simplemente ficheros de texto en un formato INI estándar (con "secciones", las cuales forman la segunda capa). Estos ficheros pueden ser modificados fácilmente usando su editor de texto favorito (como Emacs o vi), y contienen directivas leídas por el cliente para determinar qué comportamiento opcional, de varios a elegir, prefiere el usuario.

Estructura del área de configuración

La primera vez que el cliente de línea de comando **svn** es ejecutado, crea un área de configuración para el usuario. En sistemas tipo Unix, este área aparece como el directorio `.subversion` en el directorio home del usuario. En sistemas Win32, Subversion crea un directorio llamado `Subversion`, típicamente dentro del área `Application Data` del directorio de perfil del usuario (el cual, por cierto, suele ser un directorio oculto). No obstante, en esta plataforma la ubicación exacta difiere de sistema a sistema, y es dictada por el registro de Windows. [1] A lo largo del libro nos referiremos al área de configuración de cada usuario usando su nombre Unix, `.subversion`.

Además del área de configuración de cada usuario, Subversion también reconoce la existencia de un área de configuración global de sistema. Ésta le permite al administrador de sistema establecer valores por defecto para todos los usuarios en una máquina determinada. Tenga en cuenta que el área de configuración global de sistema por sí sola no dicta reglas obligatorias—los parámetros del área de configuración de cada usuario tienen prioridad a los globales, y los parámetros proporcionados en la línea de comando al programa **svn** tienen la palabra final sobre el comportamiento. En plataformas tipo Unix, el área de configuración global de sistema estará posiblemente en el directorio `/etc/subversion`; en máquinas Windows, se busca el directorio `Subversion` dentro de la ubicación común `Application Data` (de nuevo, especificada por el registro de Windows). A diferencia del caso específico de cada usuario, el programa **svn** no intentan crear un área de configuración global de sistema.

El área de configuración actualmente contiene tres ficheros—dos ficheros de configuración (`config` y `servers`), y un fichero

[1] La variable de entorno `APPDATA` apunta al área `Application Data`, así que siempre puede referirse a este directorio como `%APPDATA%\Subversion`.

`README.txt` que describe el formato INI. En el momento de su creación, los ficheros contienen los valores por defecto para cada una de las opciones soportadas por Subversion, en su mayoría con comentarios y agrupadas con descripciones textuales que indican cómo los valores de cada opción afectan al comportamiento de Subversion. Para cambiar un comportamiento dado, sólo necesita cargar el fichero de configuración adecuado en su editor de texto, y modificar el valor de la opción deseada. Si en algún momento desea recuperar los valores de configuración por defecto, puede simplemente borrar (o renombrar) su directorio de configuración y entonces ejecutar algún comando **svn** inocuo, como **svn --version**. Entonces se creará un nuevo directorio de configuración con el contenido por defecto.

El área de configuración de cada usuario también contiene una cache de datos de autenticación. El directorio `auth` agrupa un conjunto de subdirectorios que contienen trozos de información guardada usada por los varios métodos de autenticación soportados por Subversion. Este directorio es creado de tal manera que sólo su usuario tiene permiso para leer su contenido.

La configuración y el registro de Windows

Además del área de configuración habitual basada en ficheros INI, los clientes de Subversion ejecutados en plataformas Windows también pueden usar el registro de Windows para almacenar datos de configuración. Los nombres de las opciones y sus valores son iguales que en los ficheros INI. La jerarquía "fichero/sección" también se mantiene, aunque especificada de una manera ligeramente diferente—en este esquema, los ficheros y las secciones son simples niveles del árbol del registro de claves.

Subversion busca valores de configuración global de sistema bajo la clave `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion`. Por ejemplo, la opción `global-ignores`, que pertenece a la sección `miscellany` del fichero `config`, se encontraría en `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Config\Miscellany\global-ignores`. Los valores de configuración de cada usuario debería poder encontrarlos en `HKEY_CURRENT_USER\Software\Tigris.org\Subversion`.

Las opciones de configuración almacenadas en el registro son procesadas *antes* que sus versiones en ficheros de texto, por lo que sus valores son sobreescritos por lo que contengan los ficheros de configuración. En otras palabras, en un sistema Windows, la prioridad de configuración sigue el siguiente orden:

1. Opciones de línea de comando
2. Ficheros INI de cada usuario
3. Valores de registro de cada usuario
4. Ficheros INI globales de sistema
5. Valores de registro globales de sistema

Además, el registro de Windows no soporta el concepto de que algo esté "comentado". No obstante, Subversion ignorará cualquier opción cuyo nombre comience con el carácter almohadilla (#). Esto le permite comentar de forma efectiva una opción de Subversion sin tener que borrar la clave completa del registro, obviamente simplificando el proceso de recuperación de esta opción.

El cliente de línea de comando **svn** nunca intenta modificar el registro de Windows, así que no intentará crear ahí ningún área de configuración por defecto. Puede crear las claves que necesita usando el programa **REGEDIT**. Alternativamente, puede crear un fichero `.reg`, y entonces hacer doble click sobre ese fichero desde una ventana de explorador, lo cual hará que los datos sean fusionados en su registro.

Ejemplo 7.1. Fichero ejemplo de registro (.reg).

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\groups]

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\global]
"#http-proxy-host"=""
"#http-proxy-port"=""
"#http-proxy-username"=""
"#http-proxy-password"=""
"#http-proxy-exceptions"=""
"#http-timeout"="0"
"#http-compression"="yes"
"#neon-debug-mask"=""
"#ssl-authority-files"=""
"#ssl-trust-default-ca"=""
"#ssl-client-cert-file"=""
"#ssl-client-cert-password"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auth]
"#store-auth-creds"="no"

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\helpers]
"#editor-cmd"="notepad"
"#diff-cmd"=""
"#diff3-cmd"=""
"#diff3-has-program-arg"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\miscellany]
"#global-ignores"="*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#*"
"#log-encoding"=""
"#use-commit-times"=""
"#template-root"=""
"#enable-auto-props"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\tunnels]

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auto-props]
```

El ejemplo anterior muestra el contenido de un fichero `.reg` que contiene algunas de las opciones de configuración más usadas y sus valores por defecto. Tenga en cuenta la presencia tanto de opciones globales de sistema (para opciones relacionadas con proxys) como opciones de cada usuario (programas de edición y almacenamiento de claves, entre otras). Fíjese también que todas las opciones están comentadas. Sólo tiene que eliminar el carácter almohadilla (#) del comienzo de cada nombre de opción, y poner los valores que desee.

Opciones de configuración

En esta sección discutiremos las opciones específicas de parámetros de ejecución soportados actualmente por Subversion.

Servers

El fichero `servers` contiene opciones de configuración de Subversion relacionadas con las capas de red. Hay dos nombres de sección especiales en este fichero—`groups` y `global`. La sección `groups` es esencialmente una tabla de referencias cruzadas. Las claves en esta sección son nombres de otras secciones en el fichero; sus valores son *máscaras*—palabras textuales que posiblemente contienen caracteres comodines—que son comparadas contra los nombres de la máquina a la que se envían peticiones Subversion.

```
[groups]
```

```
beanie-babies = *.red-bean.com
collabnet = svn.collab.net

[beanie-babies]
…

[collabnet]
…
```

Cuando Subversion es usado en una red, intenta encontrar una coincidencia entre el nombre del servidor al que intenta llegar con un nombre de grupo de la sección `groups`. Si se encuentra tal coincidencia, Subversion busca entonces una sección en el fichero `servers` cuyo nombre coincida con el nombre del grupo. De esta sección obtendrá la configuración de red que será usada.

La sección `global` contiene la configuración que será usada para todos los servidores que no encajen con ninguna de las máscaras de la sección `groups`. Las opciones disponibles en esta sección son exactamente las mismas que aquellas válidas para otras secciones de servidor en el fichero (excepto, por supuesto, la sección especial `groups`), y son las siguientes:

`http-proxy-host`

Especifica el nombre del ordenador proxy por el cual deben pasar sus peticiones Subversion basadas en HTTP. Por defecto es un valor vacío, lo que significa que Subversion no intentará enrutar peticiones HTTP por un ordenador proxy, y que en su lugar intentará contactar la máquina destino directamente.

`http-proxy-port`

Especifica el número de puerto del ordenador proxy que debe usar. Por defecto es un valor vacío.

`http-proxy-username`

Especifica el nombre de usuario que se le facilitará al proxy. Por defecto es un valor vacío.

`http-proxy-password`

Especifica la palabra clave que se le facilitará al proxy. Por defecto es un valor vacío.

`http-timeout`

Especifica la cantidad de tiempo, en segundos, que se espera las respuestas del servidor. Si experimenta problemas con una conexión de red lenta que provoca que las operaciones de Subversion fallen por tiempo de espera agotado, debería incrementar el valor de esta opción. El valor por defecto es `0`, que indica a la librería HTTP subyacente, Neon, que use su valor tiempo de espera por defecto.

`http-compression`

Especifica si Subversion debería o no intentar comprimir las peticiones de red realizadas a servidores con soporte DAV. El valor por defecto es `yes` (aunque la compresión sólo ocurrirá si esta característica fue compilada en la capa de red). Cambie esto a `no` para desactivar la compresión, por ejemplo para depurar transmisiones por red.

`neon-debug-mask`

Esto es una máscara de bits que la librería HTTP subyacente, Neon, usa para determinar el tipo de información de depuración que debe mostrar. El valor por defecto es `0`, el cual elimina cualquier información de depuración. Para más información sobre cómo Subversion usa Neon, vea Capítulo 8, *Información para desarrolladores*.

`ssl-authority-files`

Esto es una lista delimitada con punto y coma de rutas a ficheros que contienen certificados de las autoridades de certificación (o CAs) que son aceptadas por el cliente de Subversion cuando se accede a un repositorio sobre HTTPS.

`ssl-trust-default-ca`

Modifique esta variable a `yes` si quiere que Subversion confíe de forma automática el conjunto de CAs por defecto que se distribuyen con OpenSSL.

`ssl-client-cert-file`

Si una máquina (o grupo de máquinas) requieren un certificado de cliente SSL, normalmente se le preguntará por la ruta a su certificado. Ajustando esta variable a esta misma ruta, Subversion será capaz de encontrar la certificación de su cliente automáticamente sin tener que preguntarle. No hay un lugar estándar para almacenar certificados en disco; Subversion los obtendrá de cualquier ruta que especifique.

`ssl-client-cert-password`

Si el fichero de su certificado de cliente SSL está cifrado con una frase, Subversion le preguntará por ésta cada vez que el certificado vaya a ser usado. Si considera esto un inconveniente (y no le importa almacenar la palabra clave en el fichero `servers`), entonces modifique esta variable para que contenta la clave. Ya no se le preguntará más.

Config

El fichero `config` contiene el resto de los parámetros de ejecución actualmente disponibles con Subversion, aquellos no relacionados con la conexión a red. Sólo hay un par de opciones en uso por ahora, pero de nuevo están agrupadas en secciones pues se esperan adiciones futuras.

La sección `auth` contiene parámetros relacionados con autenticación y autorización de Subversion contra un repositorio. Éstos son:

`store-auth-creds`

Esto indica a Subversion si desea mantener una caché o no de las credenciales de autenticación proporcionadas por el usuario en respuesta a demandas de autenticación por parte del servidor. El valor por defecto es `yes`. Cambie esto a `no` para desactivar la caché de credenciales en disco. Puede reemplazar esta opción en invocaciones individuales del comando **svn** usando la opción `--no-auth-cache` como parámetro (para aquellos subcomandos que la soportan). Para más información, vea “Client Credentials Caching”.

La sección `helpers` controla qué aplicaciones externas son usadas por Subversion para realizar ciertas tareas. Las opciones válidas en esta sección son:

`editor-cmd`

Especifica el programa que Subversion usará para solicitar al usuario el mensaje del informe de cambios cuando envía cambios al repositorio, como por ejemplo al usar **svn commit** sin las opciones `--message` (`-m`) o `--file` (`-F`). Este programa también se usa con el comando **svn propedit**—un fichero temporal es rellenado con los valores actuales de la propiedad que el usuario desea modificar, y los cambios toman forma en el programa editor (vea “Propiedades”). El valor por defecto de esta opción es vacío. Si la opción no está activada, Subversion comprobará las variables de entorno `SVN_EDITOR`, `VISUAL`, y `EDITOR` (en ese orden) para encontrar un comando de edición.

`diff-cmd`

Especifica una ruta absoluta a un programa de diferenciación, usado cuando Subversion genera salida “diff” (como cuando usa el comando **svn diff**). Por defecto Subversion usa una librería de diferenciación interna—activando esta opción obligará realizar esta tarea usando un programa externo.

`diff3-cmd`

Especifica una ruta absoluta a un programa de diferenciación a tres bandas. Subversion usa este programa para fusionar cambios realizados por el usuario con aquellos recibidos del repositorio. Por defecto Subversion usa una librería de diferenciación interna—activando esta opción obligará realizar esta tarea usando un programa externo.

`diff3-has-program-arg`

Este parámetro debe ajustarse a `true` si el programa especificado por la opción `diff3-cmd` acepta el parámetro de línea de comando `--diff-program`.

La sección `tunnels` le permite definir nuevos esquemas de túneles a usar con **svnserve** y conexiones cliente `svn://`. Para más detalles, vea "Autenticación y autorización SSH".

La sección `miscellany` es donde acaba todo lo que no encaja en otra parte. [2] En esta sección puede encontrar:

`global-ignores`

Cuando ejecuta el comando **svn status**, Subversion muestra un listado de ficheros y directorios no versionados junto con los versionados, anotándolos con el carácter `?` (vea "**svn status**"). A veces puede ser molesto ver elementos no versionados o no interesantes—por ejemplo, ficheros objeto resultado de la compilación de un programa— en pantalla. La opción `global-ignores` es una lista de máscaras delimitadas por espacio que describen los nombres de ficheros y directorios que Subversion no debe mostrar a no ser que estén versionados. El valor por defecto es `*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#*`.

Puede redefinir esta opción en invocaciones individuales del comando **svn status** usando la opción de línea de comando `--no-ignore`. Para más información sobre el control detallado de elementos ignorados, vea "`svn:ignore`".

`enable-auto-props`

Esto indica a Subversion que añada propiedades automáticamente en ficheros nuevos o importados. El valor por defecto es `no`, así que cambie esto a `yes` para activar auto propiedades.

La sección `auto-props` controla la capacidad del cliente de Subversion de poner automáticamente propiedades en ficheros que fueron añadidos o importados. Contiene un número de parejas clave-valor en el formato `PATRÓN = NOMBREPROPIEDAD=VALORPROPIEDAD` donde `PATRÓN` es una máscara de fichero que encaja con un grupo de ficheros y el resto de la línea es la propiedad y su valor. Múltiples coincidencias en un fichero resultarán en múltiples propiedades para ese fichero; no obstante, no hay garantía alguna de que las auto propiedades sean aplicadas en el orden en el cual fueron listadas en el fichero de configuración, así que no puede tener una regla que "redefina" otra. Puede encontrar varios ejemplos del uso de auto propiedades en el fichero `config`. Por último, no olvide cambiar `enable-auto-props` a `yes` si quiere activar las auto propiedades.

`log-encoding`

Esta variable ajusta el formato por defecto de codificación de caracteres de los informes de cambios. Es una versión permanente de la opción `--encoding` (vea "Parámetros de **svn**".) El repositorio Subversion almacena los mensajes de los informes de cambios en UTF8, y asume que su mensaje es escrito usando las locales nativas de su sistema operativo. Debería especificar una codificación diferente si sus mensajes son escritos con otra codificación.

`use-commit-times`

Normalmente los ficheros de su copia local de trabajo tienen marcas de tiempo que reflejan el último momento en que fueron tocados por cualquier proceso, ya sea su propio editor o uno de los subcomandos **svn**. Esto es normalmente conveniente para gente que desarrolla software, porque los sistemas de compilación a menudo comprueban las marcas de tiempo para decidir qué ficheros necesitan ser recompilados.

En otras situaciones, no obstante, es conveniente tener en su copia local de trabajo ficheros con marcas de tiempo que reflejan

[2] ¿Le gustan las sobras?

el último momento en el que cambiaron en el repositorio El comando **svn export** siempre pone estas “marcas temporales de última modificación” en los árboles que produce. Cambiando esta variable de configuración a `yes`, los comandos **svn checkout**, **svn update**, **svn switch**, y **svn revert** también ajustarán la marca temporal a la última modificación en el repositorio en los ficheros que manejen.

Propiedades

Ya hemos cubierto en detalle cómo Subversion almacena y recupera varias versiones de ficheros y directorios en sus repositorios. Capítulos enteros han sido dedicados a este trozo fundamental de funcionalidad proporcionada por la herramienta. Y si el soporte de versionado acabase aquí, Subversion seguiría estando completo desde una perspectiva de control de versiones. Pero no acaba aquí.

Además de versionar sus directorios y ficheros, Subversion proporciona una interfaz para añadir, modificar y eliminar meta datos versionados en cada uno de sus directorios y ficheros versionados. Nos referimos a estos meta datos como *propiedades*, y puede pensar en ellas como tablas de dos columnas que relacionan nombres de propiedades con valores arbitrarios anexos a cada elemento en su copia de trabajo local. En general, los nombres y valores de las propiedades pueden ser cualquier cosa que usted desee, con la restricción de que los nombres sean texto legible por humanos. Y la mejor parte de estas propiedades es que también son versionadas, igual que el contenido textual de sus ficheros. Puede modificar, enviar cambios al repositorio y revertir cambios sobre propiedades tan fácilmente como realiza cambios textuales. Y recibirá las modificaciones de propiedades que otras personas realicen cuando actualice su copia local de trabajo.

Otras propiedades en Subversion

Las propiedades también aparecen en otros sitios en Subversion. Igual que los ficheros y directorios pueden tener nombres y valores de propiedades arbitrarios anexos, cada revisión como un todo puede tener propiedades arbitrarias anexas. Se aplican las mismas restricciones—nombres en formato texto legibles por humanos y cualquier cosa como valor binario—con la excepción de que las propiedades de revisión no son versionadas. Vea “Propiedades no versionadas” para más información sobre estas propiedades sin versionar.

En esta sección, examinaremos la utilidad—tanto para usuarios de Subversion como para sí mismo—del soporte de propiedades. Aprenderá los subcomandos **svn** relacionados con propiedades, y cómo las modificaciones de propiedades pueden afectar a su flujo de trabajo normal con Subversion. Esperamos convencerle de que las propiedades de Subversion pueden mejorar su satisfacción con el control de versiones.

¿Por qué propiedades?

Las propiedades pueden ser adiciones muy útiles a copia de trabajo local. De hecho, Subversion usa para sí mismo propiedades para almacenar información especial, y como modo para indicar que puede ser necesario cierto procesamiento especial. Igualmente, puede usar propiedades para sus propósitos. Por supuesto, cualquier cosa que haga con propiedades puede ser realizada también usando ficheros versionados regulares, pero considere el siguiente ejemplo de uso de propiedades de Subversion.

Digamos que desea diseñar una página web que almacene muchas fotos digitales, y las muestra con descripciones y una marca de fecha. Ahora, su conjunto de fotos cambia constantemente, así que le gustaría tener la mayor parte posible de la web automatizada. Estas fotos pueden ser bastante grandes, así que como es habitual en páginas web de esta naturaleza, desea proporcionar miniaturas de las imágenes a los visitantes de su web. Puede hacer esto con ficheros tradicionales. Es decir, puede tener su `imagen123.jpg` y `imagen123-miniatura.jpg` uno al lado del otro en un directorio. O si prefiere tener los mismos nombres, puede tener las miniaturas en un directorio diferente, como `miniaturas/imagen123.jpg`. También puede almacenar sus descripciones y marcas de fechas del mismo modo, de nuevo separados del fichero original de la imagen. Pronto, su árbol de ficheros será un desorden, y crecerá en múltiplos con cada nueva foto que añada a la página web.

Ahora considere la misma situación usando las propiedades de ficheros de Subversion. Imagine tener un único fichero, `imagen123.jpg`, y varias propiedades anexas llamadas `descripcion`, `marcadetiempo`, e incluso `miniatura`. Ahora el directorio de su copia local de trabajo parece más fácil de gestionar—de hecho, parece que no hay más que ficheros de imágenes. Pe-

ro sus scripts automáticos están preparados. Saben que pueden usar **svn** (o incluso mejor, pueden usar un lenguaje de enlace con Subverion— vea "Usando lenguajes distintos de C y C++") para extraer la información adicional que su página web necesita mostrar sin tener que leer un fichero índice o jugar manipulando las rutas de ficheros.

Cómo usar las propiedades de Subversion (si decide usarlas) está en sus manos. Tal y como hemos mencionados, Subversion usa las propiedades para sus propios fines, que discutiremos más tarde en este capítulo. Pero antes, veamos cómo manipular opciones usando el programa **svn**.

Manipulando propiedades

El comando **svn** tiene la libertad de añadir o modificar propiedades de ficheros y directorios de varias maneras. Para propiedades con valores cortos, legibles por humanos, quizás la forma más simple de añadir una nueva propiedad es especificar el nombre de la propiedad y su valor en la línea de comando con el subcomando **propset**.

```
$ svn propset copyright '(c) 2003 Red-Bean Software' calc/button.c
property 'copyright' set on 'calc/button.c'
$
```

Pero hemos estado alabando la flexibilidad que Subversion ofrece para los valores de sus propiedades. Y si está planeando tener un valor de propiedad textual multi línea, o incluso binario, probablemente no quiera proporcionarlo en la línea de comando. Así que el subcomando **propset** acepta el parámetro `--file` (`-F`) para especificar el nombre de un fichero que contiene el nuevo valor de una propiedad.

```
$ svn propset license -F /path/to/LICENSE calc/button.c
property 'license' set on 'calc/button.c'
$
```

Además del comando **propset**, el programa **svn** proporciona el comando **propedit**. Este comando usa el programa de edición configurado (vea "Config") para añadir o modificar propiedades. Cuando ejecuta el comando, **svn** invoca su programa de edición sobre un fichero temporal que contiene el valor actual de la propiedad (o un fichero vacío, si está añadiendo una nueva propiedad). Entonces, simplemente modifique el valor en su programa de edición hasta que represente el nuevo valor que desea almacenar para la propiedad, guarde el fichero temporal, y salga del programa de edición. Si Subversion detecta que realmente ha modificado el fichero, aceptará esta versión como nuevo valor de la propiedad. Si sale de su programa de edición sin realizar cambios, la propiedad no será modificada.

```
$ svn propedit copyright calc/button.c  ### exit the editor without changes
No changes to property 'copyright' on 'calc/button.c'
$
```

Debemos advertirle de que al igual que con otros subcomandos de **svn**, aquellos relacionados con propiedades pueden actuar sobre varias rutas a la vez. Esto le permite modificar propiedades sobre un conjunto de ficheros con un único comando. Por ejemplo, podríamos haber hecho:

```
$ svn propset copyright '(c) 2002 Red-Bean Software' calc/*
property 'copyright' set on 'calc/Makefile'
property 'copyright' set on 'calc/button.c'
property 'copyright' set on 'calc/integer.c'
...
$
```

Todas estas adiciones y ediciones de propiedades no son realmente muy útiles si no puede recuperar fácilmente el valor almacenado en la propiedad. Así que el programa**svn** proporciona dos subcomandos para mostrar los nombres y valores de propiedades anexas a ficheros y directorios. El comando **svn proplist** mostrará un listado de los nombres de las propiedades que existen en una ru-

ta. Una vez conozca el nombre de las propiedades de un nodo, puede solicitar sus valores individualmente usando **svn propget**. Este comando, dada una ruta (o grupo de rutas) y un nombre de propiedad, imprimirá el valor de la propiedad al flujo estándar de salida.

```
$ svn proplist calc/button.c
Properties on 'calc/button.c':
  copyright
  license
$ svn propget copyright calc/button.c
(c) 2003 Red-Bean Software
```

Incluso existe una variación del comando **proplist** que mostrará tanto el nombre de todas las propiedad como su valor command. Simplemente use la opción `--verbose` (`-v`).

```
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
  license : ============================================================
Copyright (c) 2003 Red-Bean Software.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions, and the recipe for Fitz's famous
red-beans-and-rice.
…
```

El último subcomando que trata con propiedades es **propdel**. Dado que Subversion le permite almacenar propiedades con valores vacíos, no puede eliminar una propiedad usando **propedit** o **propset**. Por ejemplo, este comando *no* le proporcionará el efecto deseado:

```
$ svn propset license '' calc/button.c
property 'license' set on 'calc/button.c'
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
  license :
$
```

Debe usar el comando **propdel** para eliminar las propiedades por completo. La sintaxis es similar a la de otros comandos de propiedades:

```
$ svn propdel license calc/button.c
property 'license' deleted from ''.
$ svn proplist --verbose calc/button.c
Properties on 'calc/button.c':
  copyright : (c) 2003 Red-Bean Software
$
```

Ahora que está familiarizado con todos los subcomandos de **svn** relacionados con propiedades, veamos cómo las modificaciones de propiedades afectan al flujo de trabajo habitual de Subversion. Tal y como mencionamos anteriormente, las propiedades de ficheros y directorios están versionadas, igual que los contenidos de los ficheros. Como resultado, Subversion proporciona las mismas oportunidades de fusionar—ya sea de manera limpia o resolviendo conflictos—las modificaciones sobre propiedades de otra

persona con las suyas.

Modificando propiedades de revisiones

¿Se acuerda de esas propiedades de revisiones sin versionar? Puede modificarlas también con el programa **svn**. Simplemente añada el parámetro `--revprop` a la línea de comando, y especifique la revisión de la propiedad que desea modificar. Dado que las revisiones son globales, no necesita especificar una ruta en este caso mientras esté posicionado en la copia local de trabajo del repositorio cuya propiedad de revisión desea modificar. Por ejemplo, quizás desee reemplazar el mensaje del informe de cambios de una revisión existente. [3]

```
$ svn propset svn:log '* button.c: Fix a compiler warning.' -r11 --revprop
property 'svn:log' set on repository revision '11'
$
```

Tenga en cuenta que la capacidad para modificar estas propiedades no versionadas debe ser explícitamente añadida al repositorio por el administrador (vea "Scripts de enganche"). Dado que las propiedades no son versionadas, corre el riesgo de perder información si no es cuidadoso con sus modificaciones. El administrador del repositorio puede configurar métodos para evitar estas pérdidas, y por defecto, la modificación de propiedades no versionadas está desactivada.

Al igual que con los contenidos de sus ficheros, sus cambios sobre propiedades son modificaciones locales, únicamente convertidas en permanentes cuando las envía al repositorio con **svn commit**. También puede deshacer fácilmente sus cambios—el comando **svn revert** recuperará sus ficheros y directorios a su estado no modificado, contenido, propiedades, y todo lo demás. Además, puede recibir información interesante sobre el estado de las propiedades de sus ficheros y directorios usando los comandos **svn status** y **svn diff**.

```
$ svn status calc/button.c
 M     calc/button.c
$ svn diff calc/button.c
Property changes on: calc/button.c
___________________________________________________________________
Name: copyright
   + (c) 2003 Red-Bean Software

$
```

Fíjese cómo el subcomando **status** muestra una `M` en la segunda columna en lugar de la primera. Esto es porque hemos modificado propiedades de `calc/button.c`, pero no hemos modificado su contenido. De haber cambiado ambos, habríamos visto también una `M` en la primera columna (vea "**svn status**").

Conflictos de propiedades

Al igual que con el contenido de ficheros, las modificaciones locales de propiedades pueden entrar en conflicto con cambios enviados al servidor por otra persona. Si actualiza el directorio de su copia local de trabajo y recibe cambios de propiedades sobre un recurso versionado que chocan con los suyos, Subversion indicará que el recurso está en estado de conflicto.

```
% svn update calc
M  calc/Makefile.in
 C calc/button.c
Updated to revision 143.
$
```

[3] Corregir faltas ortográficas, errores gramaticales, y "simplemente-cosas-incorrectas" en el mensaje del informe de cambios es quizás el caso de uso más habitual para la opción `--revprop`.

Subversion también creará, en el mismo directorio que el recurso en conflicto, un fichero con la extensión `.prej` que contiene los detalles del conflicto. Debería examinar el contenido de este fichero para poder decidir cómo resolver el conflicto. Hasta que el conflicto sea resuelto, verá una `C` en la segunda columna de la salida del comando **svn status** para ese recurso, y fallará cualquier intento de enviar sus modificaciones locales al servidor.

```
$ svn status calc
 C     calc/button.c
?      calc/button.c.prej
$ cat calc/button.c.prej
prop 'linecount': user set to '1256', but update set to '1301'.
$
```

Para resolver conflictos de propiedades, simplemente asegúrese de que las propiedades en conflicto contienen los valores que deberían, y entonces use el comando **svn resolved** para indicar a Subversion que ha solucionado manualmente el problema.

Quizás también se haya dado cuenta del modo no estándar usado por Subversion para mostrar las diferencias entre propiedades. Todavía puede ejecutar **svn diff** y redirigir la salida para crear un fichero parche usable. El programa **patch** ignorará los parches de propiedades—como regla general, ignora cualquier ruido que no es capaz de entender. Esto significa desafortunadamente que para aplicar completamente un parche generado por **svn diff**, cualquier modificación de propiedades deberá ser aplicada a mano.

Como puede ver, la presencia de modificaciones de propiedades no tiene efectos significativos en el flujo de trabajo típico con Subversion. Sus patrones generales de actualizar su copia local, verificar el estado de sus ficheros y directorios, obtener informes sobre las modificaciones realizadas, y enviar éstas al repositorio son completamente inmunes a la presencia o ausencia de propiedades. El programa **svn** tiene algunos subcomandos adicionales para efectuar cambios de propiedades, pero esto es la única asimetría notable.

Propiedades especiales

Subversion no tiene reglas particulares sobre propiedades—puede usarlas para cualquier fin. Subversion sólo le pide que no use nombres de propiedades que comiencen con el prefijo `svn:`. Ese es el espacio de nombres que reserva para uso propio. De hecho, Subversion define ciertas propiedades que tienen efectos mágicos sobre ficheros y directorios. En esta sección, desvelaremos el misterio, y describiremos cómo estas propiedades especiales pueden hacerle la vida un poco más fácil.

`svn:executable`

La propiedad `svn:executable` se usa para controlar de un modo semi automático el bit de permiso de ejecución de un fichero versionado. Esta propiedad no tiene valor definido—su mera presencia indica el deseo de que el bit de permiso de ejecución se mantenga activado por Subversion. Eliminar esta propiedad devolverá el control total del bit de ejecución al sistema operativo.

En muchos sistemas operativos, la capacidad de ejecutar un fichero o comando es gobernada por la presencia de un bit de permiso de ejecución. Éste suele estar desactivado por defecto, y debe ser activado de forma explícita por el usuario para cada fichero que lo requiera. En una copia de trabajo local, se crean nuevos ficheros constantemente a medida que nuevas versiones de ficheros existentes son recibidas durante una actualización. Esto significa que quizás active el bit de ejecución en un fichero, entonces actualice su copia de trabajo, y si ese fichero fue modificado como parte de la actualización, su bit de ejecución puede haber sido desactivado. Así que Subversion proporciona la propiedad `svn:executable` para mantener el bit de ejecución activado.

Esta propiedad no tiene efecto en sistemas de ficheros que no tienen concepto de bits de permiso de ejecución, como FAT32 y NTFS. [4] Además, aunque no tenga valores definidos, Subversion forzará el valor a `*` cuando ajuste esta propiedad. Finalmente, esta propiedad es únicamente válida en ficheros, no en directorios.

`svn:mime-type`

[4] El sistema de ficheros de Windows usa extensiones de ficheros (como `.EXE`, `.BAT`, y `.COM`) para indicar que un fichero es ejecutable.

La propiedad `svn:mime-type` tiene varios propósitos en Subversion. Aparte de ser el lugar genérico para almacenar la clasificación de extensión polivalente de correo por Internet (MIME[5]), el valor de esta propiedad determina algunas características del comportamiento de Subversion.

Por ejemplo, si la propiedad `svn:mime-type` de un fichero tiene un valor de tipo MIME no textual (generalmente, algo que no comienza con `text/`, aunque hay algunas excepciones), Subversion asumirá que el fichero contiene datos binarios—es decir, no legibles por un ser humano. Uno de los beneficios habitualmente proporcionado por Subversion es el fusionado contextual, basado en líneas, de los cambios recibidos del servidor durante una actualización de su fichero de la copia local. Pero para los ficheros que se considera contienen datos binarios, no existe el concepto de una "línea". Así que para esos ficheros, Subversion no intenta realizar un fusionado contextual durante la actualización. En su lugar, siempre que tenga un fichero binario en su copia local que esté siendo actualizado, su fichero es renombrado con la extensión `.orig`, y entonces Subversion almacena un nuevo fichero que contiene los cambios recibidos durante la actualización, pero sin sus modificaciones locales, con el nombre de fichero original. Este comportamiento es realmente una protección para el usuario contra intentos fallidos de realizar fusionados contextuales sobre ficheros que simplemente no pueden ser fusionados contextualmente.

Además, si la propiedad `svn:mime-type` está ajustada, entonces el módulo Subversion de Apache usará su valor para rellenar la cabecera HTTP `Content-type:` cuando responda peticiones GET. Esto proporciona una pista vital sobre cómo mostrar un fichero cuando examina su repositorio con un navegador web.

`svn:ignore`

La propiedad `svn:ignore` contiene una lista de patrones de ficheros que serán excluidos por ciertas operaciones de Subversion. Quizás la propiedad especial usada con mayor frecuencia, funciona junto con el parámetro de ejecución `global-ignores` (vea "Config") para filtrar ficheros y directorios no versionados con los comandos **svn status**, **svn add**, y **svn import**.

La razón tras la propiedad `svn:ignore` es fácil de explicar. Subversion no asume que todo fichero o subdirectorio de una copia de trabajo local está destinado a ser puesto bajo control de versiones. Los recursos deben ser asignados explícitamente bajo la gestión de Subversion usando los comandos **svn add** o **svn import**. Como resultado, con frecuencia muchos recursos en una copia local de trabajo no están versionados.

Ahora, el comando **svn status** muestra como parte de su salida cada fichero o subdirectorio en la copia local de trabajo que no está filtrado por la opción `global-ignores` (o su valor por defecto). Esto se hace así para que los usuarios puedan ver si quizás han olvidado poner un recurso bajo control de versiones.

Pero Subversion es incapaz de adivinar los nombres de cada recurso que debe ser ignorado. Además, a menudo hay cosas que deberían ser ignoradas en *cada* copia local de trabajo de un repositorio particular. Obligar a cada usuario del repositorio a que añada patrones para estos recursos a sus áreas de configuración de parámetros de ejecución no sólo sería un incordio, sino que podría entrar en conflicto con las necesidades de configuración de otras copias locales de trabajo que el usuario ha obtenido.

La solución es almacenar los patrones de exclusión que son únicos del directorio donde deben ser aplicados junto con el propio directorio. Ejemplos habituales de recursos no versionados que son básicamente únicos de cada directorio, y propensos a aparecer ahí, son ficheros generados por la compilación de programas. O—para usar un ejemplo más apropiado para este libro—los ficheros HTML, PDF o PostScript generados como resultado de la conversión de los ficheros fuente XML DocBook a un formato de salida más legible.

Patrones de exclusión para usuarios de CVS

La propiedad `svn:ignore` de Subversion es muy similar en sintaxis y funcionalidad al fichero `.cvsignore` de CVS. De hecho, si está migrando una copia local de CVS a Subversion, puede migrar directamente los patrones de exclusión usando el fichero `.cvsignore` como fichero de entrada del comando **svn propset**:

```
$ svn propset svn:ignore -F .cvsignore .
property 'svn:ignore' set on '.'
$
```

[5] N.T.: "Multipurpose Internet Mail Extension" en inglés.

Existen, no obstante, algunas diferencias en el modo que CVS y Subversion manejan los patrones de exclusión. Los dos sistemas usan los patrones de exclusión en diferentes momentos, y hay algunas pequeñas discrepancias en cuanto a qué afectan estos patrones de exclusión. Además, Subversion no reconoce el uso del patrón `!` como comando para resetear la configuración y eliminar todos los patrones de exclusión.

Para este propósito, la propiedad `svn:ignore` es la solución. Su valor es un conjunto multi línea de patrones de ficheros, un patrón por línea. La propiedad se activa en el directorio donde quiere que los patrones sean aplicados. [6] Por ejemplo, digamos que obtiene la siguiente salida **svn status**:

```
$ svn status calc
 M      calc/button.c
?       calc/calculator
?       calc/data.c
?       calc/debug_log
?       calc/debug_log.1
?       calc/debug_log.2.gz
?       calc/debug_log.3.gz
```

En este ejemplo, ha realizado algunas modificaciones de propiedades sobre `button.c`, pero en su copia local de trabajo también tiene algunos ficheros sin versionar: la última versión del programa `calculator` que ha compilado del código fuente, un fichero de código fuente llamado `data.c`, y un conjunto de ficheros de depuración. Ahora, ya sabe que su sistema de compilado siempre acaba generando el programa `calculator`. [7] Y sabe que el conjunto de sus unidades de verificación siempre acaban dejando ficheros de depuración en el directorio. Estos hechos son ciertos para todas las copias de trabajo, no sólo la suya. Y sabe que no está interesado en ver estas cosas cada vez que ejecute **svn status**. Así que use **svn propedit svn:ignore calc** para añadir algunos patrones de exclusión al directorio `calc`. Por ejemplo, podría añadir lo siguiente como nuevo valor de la propiedad `svn:ignore`:

```
calculator
debug_log*
```

Tras añadir esta propiedad, ahora tendrá una modificación local de propiedad en el directorio `calc`. Pero advierta qué otras cosas son diferentes sobre la salida del comando **svn status**:

```
$ svn status
 M      calc
 M      calc/button.c
?       calc/data.c
```

¡Ahora, los despojos no entorpecen el listado! Por supuesto, esos ficheros aun están en su copia local de trabajo. Subversion únicamente no le está recordando que están presentes y no versionados. Y ahora tras haber eliminado todo el ruido trivial del listado, se encuentra con elementos más interesantes—como por ejemplo el fichero de código fuente que probablemente olvidó poner bajo control de versiones.

Si quiere ver los ficheros ignorados, puede pasar el parámetro **--no-ignore** a subversion:

```
$ svn status --no-ignore
 M      calc/button.c
I       calc/calculator
```

[6] Los patrones únicamente funcionarán en ese directorio—no serán aplicados de forma recursiva en subdirectorios.
[7] ¿No es este acaso el propósito de un sistema de compilación?

```
?      calc/data.c
I      calc/debug_log
I      calc/debug_log.1
I      calc/debug_log.2.gz
I      calc/debug_log.3.gz
```

La lista de patrones a excluir también es usada por **svn add** y **svn import**. Ambas de estas operaciones conllevan preguntar a Subversion que comience a gestionar algún conjunto de ficheros y directorios. En lugar de forzar al usuario que escoja qué ficheros de un árbol desea comenzar a versionar, Subversion usa los patrones de exclusión para determinar qué ficheros no deberían ser arrastrados bajo control de versiones durante grandes adiciones recursivas u operaciones de importación.

svn:keywords

Subversion tiene la capacidad de sustituir *palabras clave*—trozos dinámicos de información útil sobre un fichero versionado—en el contenido de los propios ficheros. Las palabras clave generalmente describen información sobre la última vez que el fichero fue modificado. Dado que esta información cambia cada vez que el fichero es modificado, y más importante aun, justo *después* de la modificación, es tarea difícil para todo proceso excepto el sistema de control de versiones, mantener los datos completamente actualizados. Si esta tarea fuese delegada en los autores humanos, la información inevitablemente quedaría desfasada.

Por ejemplo, digamos que posee un documento en el cual le gustaría mostrar la última fecha en la que fue modificado. Podría obligar a cada autor de ese documento a que, justo antes de enviar sus cambios al repositorio, también modifiquen la parte del documento que describe la última fecha de modificación. Pero tarde o temprano, alguien se olvidará de hacer eso. En su lugar, indique a Subversion que realice la sustitución de la palabra clave `LastChangedDate`. Puede controlar el lugar donde figura esta palabra clave colocando una *marca de palabra clave* en el lugar apropiado del fichero. Esta marca no es más que una cadena de texto formateada como `$`*`PalabraClave`*`$`.

Subversion una lista de palabras clave disponibles para ser sustituidas. Esta lista contiene las cinco siguientes palabras clave, algunas de las cuales tienen sinónimos más cortos que también puede usar:

`LastChangedDate`

> Esta palabra clave describe el último momento en que el fichero fue modificado en el repositorio, y tiene el aspecto `$LastChangedDate: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002) $`. Puede ser abreviada como `Date`.

`LastChangedRevision`

> Esta palabra clave describe la última revisión conocida en la que este fichero fue modificado en el repositorio, y tiene el aspecto `$LastChangedRevision: 144 $`. Puede ser abreviada como `Revision` o `Rev`.

`LastChangedBy`

> Esta palabra clave describe el último usuario conocido que modificó este fichero en el repositorio, y tiene el aspecto `$LastChangedBy: juan $`. Puede ser abreviada como `Author`.

`HeadURL`

> Esta palabra clave describe la URL completa a la última versión de este fichero en el repositorio, y tiene el aspecto `$HeadURL: http://svn.collab.net/repos/trunk/README $`. Puede ser abreviada como `URL`.

`Id`

> Esta palabra clave es una combinación comprimida de las otras palabras clave. Su sustitución tiene el aspecto `$Id: calc.c 148 2002-07-28 21:30:43Z carmen $`, y su interpretación indica que el fichero `calc.c` fue modificado por última vez en la revisión 148 de la tarde del 28 de Julio del 2002 por el usuario `carmen`.

Añadir únicamente la marca con la palabra clave a su fichero no hace nada especial. Subversion nunca intentará realizar sustituciones textuales en sus ficheros a no ser que usted lo pida explícitamente. Después de todo, quizás esté escribiendo un documento [8] sobre cómo usar palabras clave, ¡y no desea que Subversion sustituya sus bonitos ejemplos de marcas de palabras claves no sustituidas!

Para indicar a Subversion si desea o no sustituir palabras clave en un fichero particular, de nuevo volvemos a usar los subcomandos relacionados con propiedades. La propiedad `svn:keywords`, una vez activada en un fichero versionado, controla qué palabras clave serán sustituidas. El valor es una lista delimitada por espacios con las palabras clave o sinónimos de la tabla anterior.

Por ejemplo, digamos que tiene un fichero llamado `weather.txt` que tiene el siguiente aspecto:

```
Here is the latest report from the front lines.
$LastChangedDate$
$Rev$
Cumulus clouds are appearing more frequently as summer approaches.
```

Sin haber activado la propiedad `svn:keywords` en ese fichero, Subversion no hará nada especial. Ahora, activemos la sustitución de la palabra clave `LastChangedDate`.

```
$ svn propset svn:keywords "LastChangedDate Author" weather.txt
property 'svn:keywords' set on 'weather.txt'
$
```

Ahora acaba de realizar una modificación local de propiedad sobre el fichero `weather.txt`. No verá cambios en el contenido del fichero (a no ser que haya realizado algunos antes de activar la propiedad). Fíjese que el fichero contenía la marca de la palabra clave `Rev`, y no obstante no incluimos esta palabra clave en el valor de la propiedad. Subversion ignorará felizmente peticiones para sustituir palabras clave que no están presentes en el fichero, y no sustituirá palabras clave que no están presentes en el valor de la propiedad `svn:keywords`.

Palabras clave y diferencias espurias

El resultado de la sustitución de palabras clave visible por usuarios finales puede llevarle a pensar que cada versión de un fichero que use esta característica diferirá de la versión anterior en al menos el área donde fue colocada la marca de la palabra clave. No obstante, este no es el caso. Mientras se buscan modificaciones locales durante **svn diff**, y antes de transmitir esas modificaciones locales durante **svn commit**, Subversion "de-sustituye" palabras clave anteriormente sustituidas. El resultado es que las versiones del fichero que están almacenadas en el repositorio contienen únicamente las modificaciones reales que los usuarios realizan en el fichero.

Inmediatamente después de que envíe al repositorio la modificación de esta propiedad, Subversion actualizará la copia local de su fichero con el nuevo texto sustituido. En lugar de ver su marca de palabra clave `$LastChangedDate$`, verá el resultado de su sustitución. El resultado también contendrá el nombre de la palabra clave, y continuará estando rodeado por caracteres de dólar (`$`). Y tal y como predijimos, la palabra clave `Rev` no fue sustituida porque no solicitamos hacerlo.

```
Here is the latest report from the front lines.
$LastChangedDate: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002) $
$Rev$
Cumulus clouds are appearing more frequently as summer approaches.
```

Si alguna otra persona envía al repositorio cambios sobre el fichero `weather.txt`, su copia de ese fichero continuará mostrando el mismo valor de la sustitución de la palabra clave—hasta que actualice su copia de trabajo local. Durante ese instante las palabras

[8] … o quizás incluso una sección de un libro …

clave de su fichero `weather.txt` serán re-sustituidas para reflejar la información más reciente de ese fichero enviada al repositorio.

svn:eol-style

Mientras la propiedad `svn:mime-type` de un fichero versionado no indique lo contrario, Subversion asume que el fichero contiene datos legibles por seres humanos. En general, Subversion sólo usa este conocimiento para determinar si se pueden realizar ficheros diferenciales contextuales sobre el fichero. En caso contrario, para Subversion, los bytes son bytes.

Esto significa que por defecto, Subversion no presta atención al tipo de *marcas de fin de línea (EOL)* que use en sus ficheros. Desafortunadamente, diferentes sistemas operativos usan caracteres diferentes para representar los fines de línea en un fichero de texto. Por ejemplo, el carácter de fin de línea habitual usado por software en la plataforma Windows es una pareja de caracteres de control ASCII —retorno de carro (`CR`) y nueva línea (`LF`). En su lugar, el software Unix usa simplemente el carácter `LF` para indicar nueva línea.

No todas las herramientas en estos sistemas operativos están preparadas para entender ficheros en formatos que difieran del *estilo de fin de línea nativo* del sistema operativo sobre el cual están corriendo. Los resultados habituales son que los programas Unix tratan el carácter `CR` presente en ficheros Windows como caracteres regulares (normalmente mostrados como `^M`), y los programas Windows combinan todas las líneas de un fichero Unix en una línea gigante porque no encuentran la combinación de retorno de carro y nueva línea (o `CRLF`) que marca el fin de línea.

Esta sensibilidad a marcas EOL extrañas puede resultar frustrante para quienes comparten un fichero en múltiples sistemas operativos diferentes. Por ejemplo, considere un fichero de código fuente y desarrolladores que modifiquen este fichero tanto en Windows como en Unix. Si todos los desarrolladores usan herramientas que preservan el estilo de fin de línea del fichero, no ocurrirán problemas.

Pero en la práctica, muchas herramientas comunes o bien fallan al leer un fichero con marcas EOL extrañas, o bien convierten los finales de línea al estilo nativo cuando el fichero es guardado. Si lo primero es cierto para un desarrollador, tendrá que usar una utilidad de conversión externa (como por ejemplo **dos2unix** o su compañera, **unix2dos**) para preparar el fichero antes de modificarlo. En el segundo caso no hace falta preparación adicional. ¡Pero en ambos casos el resultado es un fichero que difiere del origen al en todas y cada una de las líneas! Antes de enviar sus cambios al repositorio, el usuario tiene dos opciones. O bien usa una utilidad de conversión para recuperar el estilo de línea que tenía el fichero antes de ser modificado. O, simplemente puede enviar sus cambios—con nuevas marcas EOL y todo.

El resultado de este tipo de escenarios incluyen tiempo perdido en modificaciones innecesarias sobre ficheros enviados al repositorio. Perder tiempo ya es doloroso. Pero cuando un cambio modifica toda línea del fichero, esto complica el trabajo de determinar qué lineas realmente cambiaron de un modo poco trivial . ¿Fue el fallo realmente corregido? ¿En qué línea se introdujo un error de sintaxis?

La solución a este problema es la propiedad `svn:eol-style`. Cuando se ajusta esta propiedad a un valor válido, Subversion la usa para determinar si tiene que realizar un proceso especial sobre el fichero para que los finales de línea del fichero no bailen como un flip-flop con cada cambio enviado al repositorio proveniente de un sistema operativo diferente. Los valores válidos son:

`native`

Esto provoca que el fichero contenta marcas EOL nativas del sistema operativo en que fue ejecutado Subversion. En otras palabras. si un usuario de Windows obtiene una copia local de un fichero cuya propiedad `svn:eol-style` contenga el valor `native`, ese fichero contendrá marcas EOL `CRLF`. Un usuario de Unix que obtenga una copia local del mismo fichero verá marcas EOL `LF`.

Tenga en cuenta que Subversion en realidad almacenará el fichero en el repositorio usando marcas EOL `LF` normalizadas sin importar el sistema operativo. No obstante, esto es básicamente transparente para el usuario.

`CRLF`

Esto provoca que el fichero contenga la secuencia `CRLF` como marcas EOL, sin importar el sistema operativo usado.

`LF`

Esto provoca que el fichero contenga el carácter `LF` como marca EOL, sin importar el sistema operativo usado.

`CR`

Esto provoca que el fichero contenga el carácter `CR` como marca EOL, sin importar el sistema operativo usado. Este estilo de fin de línea no es muy habitual. Fue usado en plataformas Macintosh antiguas (en las cuales Subversion ni si quiera funciona).

`svn:externals`

La propiedad `svn:externals` contiene instrucciones para Subversion de poblar el directorio con una o más copias locales de trabajo de Subversion. Para más información sobre esta propiedad y sus usos, vea “Repositorios externos”.

Ajuste automático de propiedades

Las propiedades son una poderosa característica de Subversion, actuando como componentes clave de muchas características de Subversion descritas en este y otros capítulos—soporte de diferenciado de ficheros y fusión textual, sustitución de palabras clave, traducción de fines de línea, etc. Pero para obtener por completo sus beneficios, debe ajustarlas en los ficheros y directorios adecuados. Desafortunadamente, este paso puede ser fácilmente olvidado en la rutina, especialmente dado que olvidar ajustar una propiedad normalmente no se traduce en una condición de error obvia (al menos si la compara, con digamos, olvidar añadir un fichero al repositorio). Para ayudar a sus propiedades a que sean aplicadas en los lugares necesarios, Subversion proporciona un par de características simples pero útiles.

Siempre que pone un fichero bajo control de revisiones usando los comandos **svn add** o **svn import**, Subversion ejecuta una heurística básica para determinar si el fichero se compone de contenido legible o no legible por un ser humano. Si decide lo último, Subversion ajustará automáticamente la propiedad `svn:mime-type` de ese fichero a `application/octet-stream` (el tipo MIME genérico “esto es un grupo de bytes”). Por supuesto, si Subversion no acierta, puede ajustar la propiedad `svn:mime-type` a otro valor más concreto—quizás `image/png` o `application/x-shockwave-flash`—siempre puede eliminar o modificar esta propiedad.

Subversion también proporciona la característica de auto propiedades, las cuales le permiten crear una relación de patrones de ficheros a nombres y valores de propiedades. Estas relaciones son creadas en su área de parámetros de ejecución. De nuevo, éstos afectan a adiciones e importaciones, y no sólo pueden sobreescribir cualquier tipo de decisión MIME por defecto tomada por Subversion, también pueden ajustar otras propiedades propias o de Subversion. Por ejemplo, podría crear una relación que dice que cada vez que añada ficheros JPEG—aquellos que coincidan con el patrón `*.jpg`—Subversion debería ajustar automáticamente la propiedad `svn:mime-type` de esos ficheros a `image/jpeg`. O quizás cualquier fichero que coincida con `*.cpp` debería tener la propiedad `svn:eol-style` ajustada a `native`, y `svn:keywords` ajustada a `Id`. El soporte de propiedades automáticas es quizás la herramienta de Subversion más útil en cuanto a propiedades. Vea “Config” para más información sobre cómo configurar esta característica.

Repositorios externos

A veces resulta útil construir una copia de trabajo local compuesta a partir de varias. Por ejemplo, podría querer que diferentes subdirectorios provengan de repositorios completamente diferentes. Decididamente puede configurar tal escenario a mano—usando **svn checkout** para crear una copia local con la estructura anidada que intenta alcanzar. Pero si esta estructura es importante para cualquiera que use el repositorio, cualquier otro usuario necesitará realizar las mismas operaciones de obtención de copias locales que usted hizo.

Afortunadamente, Subversion proporciona soporte de *definiciones externas*. Una definición externa es la relación de un directorio local con la URL—y posiblemente una revisión particular—de un recurso versionado. En Subversion, declara definiciones externas en grupos usando la propiedad `svn:externals`. Puede crear o modificar esta propiedad usando los comandos **svn propset** o **svn propedit** (vea “¿Por qué propiedades?”). Puede ser ajustada en un directorio versionado, y su valor es una tabla multi línea de subdirectorios (relativos al directorio versionado sobre el cual se pone la propiedad) y URLs de repositorios Subversion absolutas, completamente cualificadas.

```
$ svn propget svn:externals calc
third-party/sounds             http://sounds.red-bean.com/repos
third-party/skins              http://skins.red-bean.com/repositories/skinproj
third-party/skins/toolkit -r21 http://svn.red-bean.com/repos/skin-maker
```

La conveniencia de la propiedad `svn:externals` es que una vez ha sido ajustada en un directorio versionado, cualquiera que obtenga una copia local de trabajo de ese directorio, también obtendrá el beneficio de la definición externa. En otras palabras, una vez que una persona ha realizado el esfuerzo de definir esa estructura anidada de direcciones de repositorios, nadie más tiene que volver a repetirlo—ya que Subversion, a la hora de obtener una copia local, también obtendrá una copia de los repositorios externos.

Fíjese en la definición externa del ejemplo anterior. Cuando alguien obtenga la copia local del directorio `calc`, Subversion continuará obteniendo copias locales de los elementos indicados por las definiciones externas.

```
$ svn checkout http://svn.example.com/repos/calc
A  calc
A  calc/Makefile
A  calc/integer.c
A  calc/button.c
Checked out revision 148.

Fetching external item into calc/third-party/sounds
A  calc/third-party/sounds/ding.ogg
A  calc/third-party/sounds/dong.ogg
A  calc/third-party/sounds/clang.ogg
…
A  calc/third-party/sounds/bang.ogg
A  calc/third-party/sounds/twang.ogg
Checked out revision 14.

Fetching external item into calc/third-party/skins
…
```

Si necesita cambiar las definiciones externas, puede hacerlo usando los subcomandos habituales de modificación de propiedades. Cuando envíe un cambio al repositorio de la propiedad `svn:externals`, Subversion sincronizará las copias locales obtenidas contra las definiciones externas la siguiente vez que ejecute **svn update**. Lo mismo ocurrirá cuando otras personas actualicen sus copias locales y obtengan sus cambios sobre las definiciones externas.

El comando **svn status** también reconoce definiciones externas, mostrando el código de estado `X` para los subdirectorios disjuntos en los cuales se almacenarán las copias locales, entrando después de manera recursiva en estos directorios para mostrar el estado de los propios elementos externos.

No obstante, el soporte de definiciones externas que hay hoy en Subversion puede ser ligeramente engañoso. Las copias locales creadas vía definición externa todavía están desconectadas de la copia local de trabajo primaria (en cuyos directorios versionados ajustó la propiedad `svn:externals`). Y Subversion todavía funciona solamente sobre copias locales no disjuntas. Así que, por ejemplo, si quiere enviar cambios al repositorio realizados en una o más de esas copias locales externas, debe ejecutar el comando **svn commit** de forma explícita en esas copias locales—enviar los cambios de la copia local de trabajo primaria no recurrirá en las copias externas.

Además, dado que las propias definiciones son URLs absolutas, mover o copiar el directorio sobre el cual están fijadas no afectará a aquello que se obtiene como copia externa (aunque, claro está, el directorio local relativo indicado será movido con el directorio renombrado). Esto puede ser confuso—incluso frustrante—en ciertas situaciones. Por ejemplo, si usa la definición externa sobre un directorio en su línea de desarrollo principal (`/trunk`) apuntando a otras áreas de la misma línea, y entonces usa **svn copy** para crear una rama de esa línea en `/branches/my-branch`, la definición externa sobre los elementos en su nueva rama todavía se referirán a los recursos versionados en `/trunk`. Además, tenga en cuenta que si necesita reubicar su copia local de trabajo (usando **svn switch --relocate**), las definiciones externas *no* serán reubicadas.

Ramas de proveedores

En el caso especial del desarrollo de software, los datos que mantiene bajo control de versiones están a menudo relacionados con, o quizás dependan de, los datos de terceros. Generalmente, las necesidades de su proyecto dictarán que debe permanecer lo más actualizado posible con los datos proporcionados por esa entidad externa sin sacrificar la estabilidad de su propio proyecto. Este escenario ocurre constantemente—siempre que la información generada por un grupo de personas tiene un efecto directo en aquello generado por otro grupo.

Por ejemplo, desarrolladores de software podrían estar trabajando en una aplicación que hace uso de una librería externa. Subversion tiene justamente este tipo de relación con la Apache Portable Runtime library (vea “La librería Apache Portable Runtime”). El código fuente de Subversion depende de la librería APR para sus necesidades de portabilidad. En fases iniciales del desarrollo de Subversion, el proyecto seguía de cerca los cambios de la API de la APR, siempre manteniéndose en la “última versión” recién salida del horno. Ahora que tanto APR como Subversion han madurado, Subversion sólo intenta sincronizarse con la API de APR en momentos bien verificados, versiones estables públicas.

Ahora, si su proyecto depende de la información de otros, hay varios modos que podría intentar para sincronizar esa información con la suya. La más dolorosa, sería ordenar verbalmente, o por escrito, instrucciones a todos los contribuyentes de su proyecto, indicándoles que se aseguren de tener las versiones específicas de esa información externa que su proyecto necesita. Si la información externa se mantiene en un repositorio Subversion, podría usar las definiciones externas de Subversion para “fijar” de forma efectiva versiones específicas de esa información en alguno de los directorios de su copia local (vea “Repositorios externos”).

Pero a veces necesita mantener modificaciones propias a los datos de terceros en su propio sistema de control de versiones. Volviendo al ejemplo de desarrollo de software, los programadores podrían necesitar hacer modificaciones a esa librería externa para sus propios propósitos. Estas modificaciones podrían incluir nueva funcionalidad o correcciones de fallos, mantenidos internamente sólo hasta que se conviertan en parte de un lanzamiento oficial de esa librería externa. O los cambios quizás nunca sean enviados a los autores de la librería, y existen solamente con el propósito de satisfacer las necesidades de sus desarrolladores como modificaciones personalizadas.

Ahora se encuentra en una situación interesante. Su proyecto podría almacenar sus modificaciones propias a los datos de terceros de algún modo disjunto, como usando ficheros parche o completas versiones alternativas de ficheros y directorios. Pero estos métodos se convierten rápidamente en problemas de mantenimiento, requiriendo de algún mecanismo que aplique sus cambios personales a los datos de terceros, sin olvidar la necesidad de regenerar esos cambios con cada versión sucesiva de los datos de terceros a los que sigue la pista.

La solución a este problema es usar *ramas de proveedor*. Una rama de proveedor es un árbol de directorios en su propio sistema de control de versiones que contiene información proporcionada por una entidad externa, o proveedor. Cada versión de los datos del proveedor que decide absorber en su proyecto es llamada *hito de proveedor*.

Las ramas de proveedor proporcionan dos beneficios clave. Primero, al almacenar el hito de proveedor actualmente soportado en su propio sistema de control de versiones, los miembros de su proyecto nunca tendrán que preguntarse si poseen la versión correcta de los datos del proveedor. Simplemente obtienen la versión correcta como parte de su actualización habitual de sus copias locales de trabajo. Segundo, ya que los datos viven en su propio repositorio Subversion, puede almacenar cambios personalizados directamente en el mismo lugar—ya no tiene necesidad de crear un método automático (o peor aún, manual) para proporcionar sus modificaciones.

Procedimiento general de gestión de ramas de proveedor

Gestionar ramas de vendedor generalmente funciona de esta manera. Primero crea un directorio en la raíz de su jerarquía (como por ejemplo `/vendor`) para almacenar las ramas de proveedor. Entonces importa el código de terceros en un subdirectorio de ese directorio principal. Luego copia ese subdirectorio en su rama principal de desarrollo (por ejemplo, `/trunk`) en un lugar apropiado. Siempre realiza cambios locales en la rama de desarrollo principal. Con cada nueva versión del código que está siguiendo, lo incluye en la rama de proveedor y fusiona los cambios en `/trunk`, resolviendo cualquier conflicto entre sus cambios locales y los cambios oficiales.

Quizás un ejemplo ayude a aclarar este algoritmo. Usaremos un escenario donde su equipo de desarrollo está creando un programa calculador que enlaza contra una librería externa de aritmética de números complejos, libcomplex. Comenzaremos con la creación inicial de la rama de proveedor, y la importación del primer hito de proveedor. Llamaremos al directorio de nuestra rama de pro-

veedor `libcomplex`, y nuestros hitos de código irán en un subdirectorio de nuestra rama de proveedor llamado `current`. Y dado que **svn import** crea todos los directorios padre intermedios que necesita, en realidad podemos realizar ambos pasos con un único comando.

```
$ svn import /path/to/libcomplex-1.0 \
             http://svn.example.com/repos/vendor/libcomplex/current \
             -m 'importing initial 1.0 vendor drop'
…
```

Tenemos ahora la versión actual del código fuente de libcomplex en `/vendor/libcomplex/current`. Ahora, etiquetamos esa versión (vea "Etiquetas") y luego la copiamos en la rama principal de desarrollo. Nuestra copia creará un nuevo directorio llamado `libcomplex` en nuestro directorio existente de proyecto `calc`. Es en esta versión copiada de los datos del proveedor que realizaremos nuestras personalizaciones.

```
$ svn copy http://svn.example.com/repos/vendor/libcomplex/current  \
           http://svn.example.com/repos/vendor/libcomplex/1.0      \
           -m 'tagging libcomplex-1.0'
…
$ svn copy http://svn.example.com/repos/vendor/libcomplex/1.0  \
           http://svn.example.com/repos/calc/libcomplex        \
           -m 'bringing libcomplex-1.0 into the main branch'
…
```

Ahora obtenemos una copia local de la rama principal de nuestro proyecto—que ahora incluye una copia del primer hito de proveedor—y comenzamos a personalizar el código de libcomplex. Antes de darnos cuenta, nuestra versión modificada de libcomplex está completamente integrada en nuestro programa calculador. [9]

Unas pocas semanas después, los desarrolladores de libcomplex lanzan una nueva versión de su librería—la versión 1.1—la cual contiene algunas características y funcionalidades que realmente deseamos. Nos gustaría actualizarnos a esta nueva versión, pero sin perder las personalizaciones que realizamos sobre la versión anterior. Lo que esencialmente nos gustaría hacer, es reemplazar nuestra versión de línea base de libcomplex 1.0 con una copia de libcomplex 1.1, y entonces reaplicar las modificaciones propias que anteriormente hicimos sobre la librería antes de la nueva versión. Pero en realidad nos acercaremos al problema desde otra dirección, aplicando los cambios realizados sobre libcomplex entre las versiones 1.0 y 1.1 sobre nuestra propia copia modificada.

Para realizar esta actualización, obtenemos una copia local de nuestra rama de proveedor, y reemplazamos el código del directorio `current` con el nuevo código fuente de libcomplex 1.1. Literalmente copiamos nuevos ficheros sobre los existentes, quizás expandiendo el fichero comprimido del distribución de libcomplex 1.1 sobre nuestros ficheros y directorios. El objetivo aquí es que el directorio `current` solamente contenta el código de libcomplex 1.1, y asegurarnos que todo ese código fuente está bajo control de versiones. Oh, y queremos realizar esto con la menor perturbación posible sobre el historial del control de versiones.

Tras reemplazar el código de la versión 1.0 con el de la 1.1, **svn status** mostrará los ficheros con modificaciones locales, junto con quizás algunos ficheros no versionados o ausentes. Si realizamos lo que debíamos realizar, los ficheros no versionados son aquellos ficheros nuevos introducidos en la versión 1.1 de libcomplex—ejecutamos **svn add** sobre ellos para ponerlos bajo control de versiones. Los ficheros ausentes son ficheros que existían en la 1.0 pero no en la 1.1, y sobre éstos usamos **svn remove**. Finalmente, una vez que nuestra copia local de `current` contiene únicamente el código de libcomplex 1.1, enviamos al repositorio los cambios realizados para que tenga este aspecto.

Nuestra rama `current` ahora contiene el hito de proveedor. Etiquetamos la nueva versión (del mismo modo que etiquetamos el anterior hito de proveedor como la versión 1.0), y entonces fusionamos las diferencias entre las etiquetas de la versión anterior y la actual en nuestra rama principal de desarrollo.

```
$ cd working-copies/calc
$ svn merge http://svn.example.com/repos/vendor/libcomplex/1.0      \
```

[9] ¡Y por su puesto, completamente libre de fallos!

```
            http://svn.example.com/repos/vendor/libcomplex/current  \
            libcomplex
… # resolve all the conflicts between their changes and our changes
$ svn commit -m 'merging libcomplex-1.1 into the main branch'
…
```

En el caso de uso trivial, la nueva versión de nuestra herramienta de terceros sería, desde un punto de vista de ficheros y directorios, justo igual que nuestra versión anterior. Ninguno de los ficheros de código fuente de libcomplex habría sido borrado, renombrado o movido a una ubicación diferente—la nueva versión solamente tendría modificaciones textuales contra la anterior. En un mundo perfecto, nuestras modificaciones serían aplicadas limpiamente sobre la nueva versión de la librería, absolutamente sin complicaciones o conflictos.

Pero las cosas no son siempre tan simples, y de hecho es bastante habitual que ficheros de código fuente sean desplazados entre versiones de un software. Esto complica el proceso de asegurarnos que nuestras modificaciones son todavía válidas para la nueva versión del código, y puede degenerar rápidamente en la situación donde tenemos que recrear manualmente nuestras personalizaciones para la nueva versión. Una vez Subversion conoce la historia de un fichero de código fuente determinado—incluyendo todas sus ubicaciones previas—el proceso de fusionar la nueva versión de la librería es bastante simple. Pero somos responsables de indicar a Subversion cómo ha cambiado la estructura de ficheros de un hito de proveedor a otro.

svn_load_dirs.pl

Los hitos de proveedor que conllevan algo más que algunos borrados, adiciones o movimientos de ficheros complican el proceso de actualizarse a cada versión sucesiva de los datos de terceros. Por lo que Subversion proporciona el script **svn_load_dirs.pl** para asistirle en este proceso. Este script automatiza los procesos de importado anteriormente mencionados en el proceso de gestión de la rama del proveedor para minimizar el número posible de errores. Todavía será responsable de usar los comandos de fusionado para fusionar las nuevas versiones de los datos de terceros en su rama de desarrollo principal, pero **svn_load_dirs.pl** puede ayudarle a llegar a esta fase más rápido y con facilidad.

En resumen, **svn_load_dirs.pl** es una mejora a **svn import** que tiene ciertas características importantes:

- Puede ser ejecutado en cualquier momento para transformar un directorio existente del repositorio para que sea una réplica de un directorio externo, realizando todas las operaciones de adición y borrado necesarias, y también opcionalmente haciendo algunas operaciones de renombrado.
- Se encarga de una serie de operaciones complicadas entre las cuales Subversion requiere como paso intermedio enviar cambios al repositorio— como antes de renombrar un fichero o directorio dos veces.
- Opcionalmente etiquetará el nuevo directorio importado.
- Opcionalmente añadirá propiedades arbitrarias a ficheros y directorios que coincidan con una expresión regular.

svn_load_dirs.pl recibe tres parámetros obligatorios. El primer parámetro es la URL al directorio base de Subversion en el que se realizará el trabajo. Este parámetro es seguido por la URL—relativa respecto al primer argumento—en la cual se importará el hito de proveedor actual. Finalmente, el tercer parámetro es el directorio local que desea importar. Usando nuestro ejemplo anterior, una ejecución típica de **svn_load_dirs.pl** podría ser:

```
$ svn_load_dirs.pl http://svn.example.com/repos/vendor/libcomplex \
                   current                                        \
                   /path/to/libcomplex-1.1
…
```

Puede indicar que desearía que **svn_load_dirs.pl** etiquete el nuevo hito de proveedor pasando la línea de comando `-t` e indicando un nombre de etiqueta. Esta etiqueta es otra URL relativa al primer argumento pasado al programa.

```
$ svn_load_dirs.pl -t libcomplex-1.1                                  \
                   http://svn.example.com/repos/vendor/libcomplex \
                   current                                            \
                   /path/to/libcomplex-1.1
…
```

Cuando ejecuta **svn_load_dirs.pl**, examina el contenido de su hito de proveedor "actual", y lo compara con el hito de proveedor propuesto. En el caso trivial, no habrá ficheros que existan en una versión pero no en la otra, así que el script realizará la nueva importación sin incidentes. No obstante, si hay discrepancias en la estructura de ficheros entre las versiones, **svn_load_dirs.pl** le preguntará cómo desea resolver esas diferencias. Por ejemplo, tendrá la oportunidad de decirle al script que sabe que el fichero `math.c` en la versión 1.0 de libcomplex fue renombrado como `arithmetic.c` en libcomplex 1.1. Cualquier discrepancia no explicada como renombrado será tratada como una adición o borrado normal.

El script también acepta un fichero de configuración separado para ajustar propiedades sobre ficheros y directorios que coincidan con una expresión regular y que vayan a ser *añadidos* al repositorio. Este fichero de configuración se le indica a **svn_load_dirs.pl** usando la opción de línea de comando `-p`. Cada línea del fichero de configuración es un conjunto de dos o cuatro valores separados por espacios en blanco: una expresión regular estilo Perl que debe coincidir con una ruta, una palabra de control (ya sea `break` o `cont`), y opcionalmente, un nombre de propiedad y su valor.

```
\.png$              break   svn:mime-type   image/png
\.jpe?g$            break   svn:mime-type   image/jpeg
\.m3u$              cont    svn:mime-type   audio/x-mpegurl
\.m3u$              break   svn:eol-style   LF
.*                  break   svn:eol-style   native
```

Para cada ruta añadida, se aplicarán en orden los cambios de propiedades configurados para las rutas que coincidan con las expresiones regulares, a no ser que la especificación de control sea `break` (lo cual significa que no deben aplicarse más cambios de propiedades a esa ruta). Si la especificación de control es `cont`—una abreviación de `continue`—entonces se seguirá buscando coincidencias con el patrón de la siguiente línea del fichero de configuración.

Cualquier carácter de espaciado en la expresión regular, nombre de la propiedad, o valor de la propiedad deberá ser rodeado por caracteres de comillas simples o dobles. Puede escapar los caracteres de comillas que no son usados para envolver caracteres de espaciado precediéndolos con un carácter de contrabarra (\). Las contrabarras sólo escapan las comillas cuando se procesa el fichero de configuración, así que no proteja ningún otro carácter excepto lo necesario para la expresión regular.

Capítulo 8. Información para desarrolladores

Subversion es un proyecto de software open-source desarrollado bajo una licencia de software estilo Apache. El proyecto está respaldado económicamente por CollabNet, Inc., una compañía de desarrollo de software con sede en California. La comunidad que se ha formado alrededor del desarrollo de Subversion siempre da la bienvenida a nuevos miembros que pueden donar su tiempo y atención al proyecto. A los voluntarios se les anima a ayudar de cualquier modo, ya sea encontrando y diagnosticando fallos, refinando el código fuente existente, o implementando por nuevas características.

Este capítulo es para aquellos que desean asistir en la evolución continua de Subversion ensuciándose las manos con el código fuente. Cubriremos algunos detalles íntimos del software, ese tipo de conocimiento técnico específico necesario por aquellos que desarrollan Subversion—o herramientas completamente nuevas basadas en las librerías de Subversion. Si no tiene previsto participar con el software a tal nivel, sientase libre de ignorar este capítulo con la certeza de que su experiencia como usuario de Subversion no se verá afectada.

Diseño de librería por capas

Subversion tiene un diseño modular, implementado como un grupo de librerías en C. Cada librería tiene un propósito e interfaz bien definidos, y la mayoría de los módulos se puede decir que existen en una de tres posibles capas—capa de repositorio, capa de acceso al repositorio (RA[1]), o capa de cliente. Examinaremos en breve estas capas, pero antes, vea un corto inventario de las librerías de Subversion en Tabla 8.1, “Un corto inventario de las librerías de Subversion”. En aras de la consistencia, nos referiremos a estas librerías por sus nombres de librería Unix sin extensión (por ejemplo: libsvn_fs, libsvn_wc, mod_dav_svn).

Tabla 8.1. Un corto inventario de las librerías de Subversion

Librería	Descripción
libsvn_client	Interfaz principal para programas cliente
libsvn_delta	Rutinas de diferenciado de árboles y texto
libsvn_fs	Librería del sistema de ficheros de Subversion
libsvn_ra	Utilidades comunes de acceso al repositorio y módulo cargador
libsvn_ra_dav	Módulo de acceso WebDav al repositorio
libsvn_ra_local	Módulo de acceso local al repositorio
libsvn_ra_svn	Módulo de acceso al repositorio mediante protocolo personalizable
libsvn_repos	Interfaz del repositorio
libsvn_subr	Miscelánea de subrutinas útiles
libsvn_wc	Librería para la gestión de la copia local de trabajo
mod_authz_svn	Módulo de autorización de Apache para acceso a repositorios Subversion vía WebDAV
mod_dav_svn	Módulo Apache para relacionar operaciones WebDAV con operaciones Subversion

El hecho de que la palabra “miscelánea” sólo aparezca una vez en Tabla 8.1, “Un corto inventario de las librerías de Subversion” es una buena señal. El equipo de desarrollo de Subversion está seriamente dedicado a que las funcionalidades vivan en la capa y librería correctas. Quizás la mayor de las ventajas del diseño modular es su falta de complejidad desde el punto de vista de un desa-

[1] N.T.: “repository access”.

rrollador. Como desarrollador, usted puede formular rápidamente este “gran mapa” que le permite marcar la ubicación de ciertas partes de la funcionalidad con relativa facilidad.

Otro gran beneficio de la modularidad es la posibilidad de reemplazar un módulo concreto con una nueva librería que implementa la misma API sin afectar al resto del código fuente. En cierto sentido, esto ya ocurre dentro de Subversion. Las librerías libsvn_ra_dav, libsvn_ra_local, y libsvn_ra_svn implementan la misma interfaz. Y las tres se comunican con la capa de repositorio;libsvn_ra_dav y libsvn_ra_svn lo hacen a través de la red, mientras que libsvn_ra_local accede directamente.

El propio cliente es una muestra de la modularidad del diseño de Subversion. A pesar de que Subversion por ahora sólo viene con un programa cliente de línea de comando, ya hay en desarrollo unos cuantos programas por parte de terceros que actúan como interfaces gráficas de Subversion. De nuevo, estos GUIs usan las mismas APIs que el cliente de línea de comando que viene de serie. La librería de Subversion libsvn_client es como una tienda de paso con la mayor parte de la funcionalidad necesaria para diseñar un cliente de Subversion (vea “Capa cliente”).

Capa de repositorio

Cuando nos referimos a la capa de repositorio de Subversion, en general estamos hablando de dos librerías—la librería de repositorio, y la librería del sistema de ficheros. Estas librerías proporcionan los mecanismos de almacenamiento e informe para las varias revisiones de sus datos bajo control de versiones. Esta capa está conectada a la capa cliente vía capa de acceso al repositorio, y son, desde el punto de vista del usuario de Subversion, las cosas “al otro lado de la línea.”

Al sistema de ficheros de Subversion se accede vía la API de libsvn_fs, y no se trata de un sistema de ficheros que uno instalaría en un sistema operativo (como ext2 de Linux o NTFS), sino un sistema de ficheros virtual. En lugar de almacenar “ficheros” y “directorios” como ficheros y directorios reales (vamos, el tipo que puede explorar usando su programa favorito de shell), usa un sistema de base de datos para sus mecanismos back-end de almacenamiento. Actualmente el sistema de base de datos que usa es la base de datos de Berkeley. [2] No obstante, en la comunidad de desarrolladores ha habido un considerable interés para que futuras versiones de Subversion tengan la capacidad para usar otros sistemas back-ends de base de datos, quizás a través de mecanismos como la Open Database Connectivity (ODBC).

La API de sistema de ficheros exportada por libsvn_fs contiene el tipo de funcionalidad que esperaría de cualquier otro API de sistema de ficheros: puede crear y borrar ficheros y directorios, copiarlos y moverlos, modificar el contenido de ficheros, y un largo etcétera. También tiene características que no son tan comunes, como la habilidad de añadir, modificar o eliminar meta datos (“propiedades”) sobre cualquier fichero o directorio. Además, el sistema de ficheros de Subversion es versionado, lo que significa que a medida que realiza cambios a su árbol de directorios, Subversion recuerda el aspecto que tenía el árbol antes de esos cambios. Y antes que los cambios anteriores. Y que los anteriores a los anteriores. Y así todo el camino de vuelta por el tiempo de versionado hasta el momento inicial en el cual comenzó a añadir cosas al sistema de ficheros.

Todas las modificaciones que haga a su árbol se realizarán en el contexto de una transacción de Subversion. Lo siguiente es una rutina general simplificada para modificar su sistema de ficheros:

1. Comience una transacción de Subversion.
2. Realice sus cambios (adiciones, borrados, modificación de propiedades, etc.).
3. Finalice su transacción.

Una vez ha finalizado su transacción, las modificaciones a su sistema de ficheros se almacenan permanentemente como artefactos históricos . Cada uno de estos ciclos genera una nueva revisión de su árbol, y cada revisión es accesible para siempre como una instantánea inmutable de “cómo estaban las cosas.”

[2] La elección de la base de datos de Berkeley trajo varias características de forma automática que Subversion necesitaba, como integridad de datos, escrituras atómicas, recuperabilidad, y copias de seguridad en caliente.

La distracción de la transacción

La noción de una transacción de Subversion puede ser confundida fácilmente con el soporte de transacción proporcionado por la base de datos subyacente, especialmente dada su proximidad al código de base de datos en libsvn_fs. Ambos tipos de transacción existen para proporcionar atomicidad y aislamiento. En otras palabras, las transacciones le permiten realizar un conjunto de acciones de una forma "todo o nada"—o bien todas las acciones del conjunto son completadas con éxito, o todas son tratadas como si *ninguna* de ellas hubiese ocurrido—y de un modo que no interfieran con otros procesos que actúan sobre los datos.

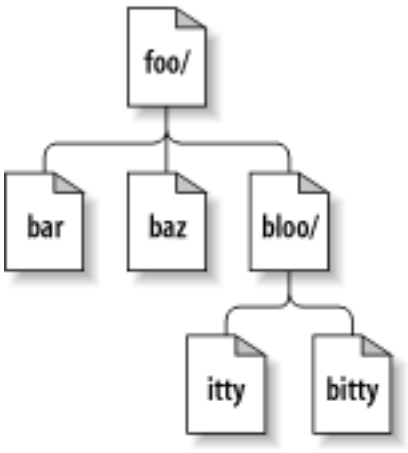
Las transacciones de base de datos generalmente engloban pequeñas operaciones relacionadas con la modificación específica de datos (como cambiar el contenido de una fila en una tabla). Las transacciones de Subversion son mayores en ámbito, englobando operaciones de alto nivel como realizar modificaciones a un conjunto de ficheros y directorios que pretenden ser almacenados como la siguiente revisión del árbol del sistema de ficheros. Si eso no es suficientemente confuso, considere lo siguiente: ¡Subversion usa una transacción de base de datos durante la creación de una transacción de Subversion (por lo que si la creación de una transacción de Subversion falla, la base de datos tendrá el mismo aspecto que antes, como si nunca hubiésemos intentado crear nada)!

Afortunadamente para los usuarios de la API del sistema de ficheros, el soporte de transacción proporcionado por el propio sistema de base de datos está oculto casi por completo (tal y como debería esperarse de un esquema de librerías correctamente modularizadas). Sólo cuando comienza a ahondar en la implementación del sistema de ficheros este tipo de cosas se vuelven visibles (o interesantes).

La mayoría de la funcionalidad proporcionada por la interfaz del sistema de ficheros se muestra como acciones que trabajan sobre una ruta del sistema de ficheros. Es decir, desde un punto de vista externo, el mecanismo principal para describir y acceder a las revisiones individuales de ficheros y directorios son las cadenas con rutas como `/foo/bar`, igual que si estuviese tratando con ficheros y directorios a través de su programa favorito de línea de comando. Añada nuevos ficheros y directorios pasando sus futuras rutas a las funciones correctas de la API. Obtenga información sobre ellos usando el mismo mecanismo.

No obstante, a diferencia de la mayoría de los sistemas de ficheros, una ruta sola no es información suficiente para identificar un fichero o directorio en Subversion. Piense que el árbol de directorios es un sistema bidimensional, donde los nodos hermanos representan una especie de movimiento izquierda-derecha, y descender en subdirectorios un movimiento hacia abajo. Figura 8.1, "Ficheros y directorios en dos dimensiones" muestra una representación típica de un árbol justo de ese modo.

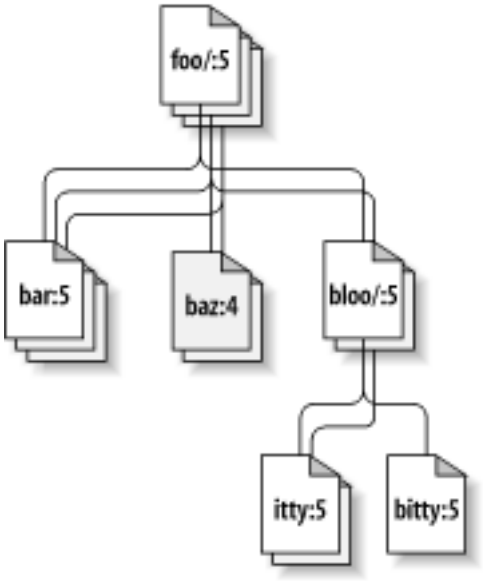
Figura 8.1. Ficheros y directorios en dos dimensiones



Por supuesto, el sistema de ficheros de Subversion tiene una genial tercera dimensión que muchos sistemas de ficheros no tienen—¡El tiempo! [3] En la interfaz del sistema de ficheros, casi toda función que tiene un parámetro de *`ruta`* espera también un pa-

rámetro *raíz*. Este parámetro svn_fs_root_t describe una revisión o una transacción de Subversion (la cual es normalmente una futura revisión), y proporciona ese contexto tridimensional necesario para entender la diferencia entre `/foo/bar` de la revisión 32, y la misma ruta tal y como existe en la revisión 98. Figura 8.2, "Versionando el tiempo—¡la tercera dimensión!" muestra la historia de versiones como una dimensión añadida al universo del sistema de ficheros de Subversion.

Figura 8.2. Versionando el tiempo—¡la tercera dimensión!



Tal y como mencionamos anteriormente, la API de libsvn_fs tiene el aspecto de cualquier otro sistema de ficheros, excepto por el hecho de tener la maravillosa capacidad de versionado. Fue diseñada para ser usada por cualquier programa interesado en un sistema de ficheros versionado. A su vez, Subversion está interesado en esa funcionalidad. Pero mientras que la API del sistema de ficheros debería ser suficiente para tener soporte básico de versionado de ficheros y directorios, Subversion quiere más—y es ahí donde entra en juego libsvn_repos.

La librería de repositorio de Subversion (libsvn_repos) es básicamente una librería envoltorio sobre la funcionalidad del sistema de ficheros. Esta librería es responsable de crear la estructura del repositorio, asegurarse de que el sistema de ficheros subyacente está inicializado, etc. Libsvn_repos también implementa un conjunto de ganchos—scripts que son ejecutados por el código de repositorio cuando se realizan ciertas tareas. Estos scripts son útiles para notificar, autorizar, o cualquier cosa deseada por el administrador del repositorio. Este tipo de funcionalidad, y otras utilidades proporcionadas por la librería de repositorio, no están estrictamente relacionadas con la implementación de un sistema de ficheros versionados, razón por la cual fue colocada en su propia librería.

Los desarrolladores que deseen usar la API de libsvn_repos se encontrarán con que no es un envoltorio completo sobre la interfaz del sistema de ficheros. Es decir, sólo ciertos eventos de categoría en el ciclo general de la actividad del sistema de ficheros están envueltos por la interfaz de repositorio. Algunos de éstos incluyen la creación y enviado de transacciones Subversion, y la modificación de propiedades de revisiones. Estos eventos particulares están envueltos por la capa de repositorio porque tienen enganches asociados con ellos. En el futuro, puede que otros eventos sean envueltos por la API de repositorio. Mientras tanto, el resto de la interacción con el sistema de ficheros continuará ocurriendo directamente vía la API de libsvn_fs.

Por ejemplo, aquí tiene un segmento de código que ilustra el uso de ambas interfaces (repositorio y sistema de ficheros) para crear una nueva revisión del sistema de ficheros en la cual se añade un directorio. Tenga en cuenta que en este ejemplo (y todos los demás del libro), la macro SVN_ERR simplemente verifica de forma booleana el código de retorno de la función que envuelve, y retorna si hay algún error.

[3] Entendemos que esto pueda ser un jarro de agua fría a todos los fans de la ciencia ficción, quienes durante mucho tiempo han considerado el tiempo como la *cuarta* dimensión, y pedimos disculpas si provocamos algún trauma emocional con nuestra aserción de una teoría diferente.

Ejemplo 8.1. Usando la capa de repositorio

```
/* Create a new directory at the path NEW_DIRECTORY in the Subversion
   repository located at REPOS_PATH.  Perform all memory allocation in
   POOL.  This function will create a new revision for the addition of
   NEW_DIRECTORY.  */
static svn_error_t *
make_new_directory (const char *repos_path,
                    const char *new_directory,
                    apr_pool_t *pool)
{
  svn_error_t *err;
  svn_repos_t *repos;
  svn_fs_t *fs;
  svn_revnum_t youngest_rev;
  svn_fs_txn_t *txn;
  svn_fs_root_t *txn_root;
  const char *conflict_str;

  /* Open the repository located at REPOS_PATH.  */
  SVN_ERR (svn_repos_open (&repos, repos_path, pool));

  /* Get a pointer to the filesystem object that is stored in
     REPOS.  */
  fs = svn_repos_fs (repos);

  /* Ask the filesystem to tell us the youngest revision that
     currently exists.  */
  SVN_ERR (svn_fs_youngest_rev (&youngest_rev, fs, pool));

  /* Begin a new transaction that is based on YOUNGEST_REV.  We are
     less likely to have our later commit rejected as conflicting if we
     always try to make our changes against a copy of the latest snapshot
     of the filesystem tree.  */
  SVN_ERR (svn_fs_begin_txn (&txn, fs, youngest_rev, pool));

  /* Now that we have started a new Subversion transaction, get a root
     object that represents that transaction.  */
  SVN_ERR (svn_fs_txn_root (&txn_root, txn, pool));

  /* Create our new directory under the transaction root, at the path
     NEW_DIRECTORY.  */
  SVN_ERR (svn_fs_make_dir (txn_root, new_directory, pool));

  /* Commit the transaction, creating a new revision of the filesystem
     which includes our added directory path.  */
  err = svn_repos_fs_commit_txn (&conflict_str, repos,
                                 &youngest_rev, txn, pool);
  if (! err)
    {
      /* No error?  Excellent!  Print a brief report of our success.  */
      printf ("Directory '%s' was successfully added as new revision "
              "'%" SVN_REVNUM_T_FMT "'.\n", new_directory, youngest_rev);
    }
  else if (err->apr_err == SVN_ERR_FS_CONFLICT)
    {
      /* Uh-oh.  Our commit failed as the result of a conflict
         (someone else seems to have made changes to the same area
         of the filesystem that we tried to modify).  Print an error
         message.  */
      printf ("A conflict occurred at path '%s' while attempting "
```

```
              "to add directory '%s' to the repository at '%s'.\n",
              conflict_str, new_directory, repos_path);
    }
  else
    {
      /* Some other error has occurred.  Print an error message.  */
      printf ("An error occurred while attempting to add directory '%s' "
              "to the repository at '%s'.\n",
              new_directory, repos_path);
    }

  /* Return the result of the attempted commit to our caller.  */
  return err;
}
```

En el segmento de código anterior, se hacen llamadas tanto a las interfaces de repositorio como de sistema de ficheros. Igualmente podríamos haber enviado la transacción usando `svn_fs_commit_txn`. Pero la API del sistema de ficheros no sabe nada sobre los mecanismos de enganche de la librería del repositorio. Si quiere que su repositorio de Subversion realice automáticamente un conjunto de tareas no relacionadas con Subversion cada vez que envía una transacción (como por ejemplo, enviar un email que describe los cambios realizados en esa transacción a su lista de correo de desarrolladores), necesita usar la versión envoltorio de la función libsvn_repos apropiada—`svn_repos_fs_commit_txn`. Esta función en realidad ejecutará primero el script de enganche `pre-commit` si existe, entonces enviará la transacción, y finalmente ejecutará el script de enganche `post-commit`. Los ganchos proporcionan un mecanismo especial de información que no pertenece a la librería del núcleo del sistema de ficheros. (Para más información sobre los ganchos de repositorio de Subversion, lea “Scripts de enganche”.)

El requisito del mecanismo de enganches es una de las razones para abstraer la librería de repositorio del código del sistema de ficheros. La API libsvn_repos proporciona algunas otras utilidades importantes para Subversion. Entre ellas está la capacidad para:

1. crear, abrir, destruir y realizar los pasos de recuperación sobre un repositorio Subversion y el sistema de ficheros incluido en el mismo.

2. describir las diferencias entre dos árboles de sistema de ficheros.

3. obtener el informe de cambios asociado con todas (o algunas) las revisiones en las cuales un conjunto de ficheros fue modificado en el sistema de ficheros.

4. generar un “volcado” legible por un humano del sistema de ficheros, una representación completa de las revisiones en el sistema de ficheros.

5. procesar el formato de ese volcado, cargando revisiones volcadas en un repositorio Subversion diferente.

A medida que Subversion continúa evolucionando, la librería de repositorio seguirá creciendo con la librería del sistema de ficheros para ofrecer una mayor funcionalidad y soporte configurable de opciones.

Capa de acceso al repositorio

Si la capa de repositorio de Subversion está “al otro lado de la línea”, la capa de acceso al repositorio está en la propia línea. Destinada a serializar los datos entre las librerías cliente y el repositorio, esta capa incluye la librería cargadora del módulo libsvn_ra, los propios módulos RA (que por ahora son libsvn_ra_dav, libsvn_ra_local, y libsvn_ra_svn), y cualquier librería adicional necesaria por uno o varios de esos módulos RA, como por ejemplo el módulo mod_dav_svn de Apache que se comunica con libsvn_ra_dav o el servidor de libsvn_ra_svn, **svnserve**.

Dado que Subversion usa URLs para identificar los recursos del repositorio, la porción de protocolo del esquema URL (normalmente `file:`, `http:`, `https:`, o `svn:`) es usada para determinar qué módulo RA se encargará de las comunicaciones. Cada módulo registra una lista de los protocolos que sabe "hablar" por lo que el cargador de RA puede, en tiempo de ejecución, determinar qué modulo usar para cada tarea concreta. Puede determinar qué módulos RA están disponibles en el cliente de línea de comando de Subversion, y qué protocolos dice soportar, ejecutando **svn --version**:

```
$ svn --version
svn, version 1.0.1 (r9023)
   compiled Mar 17 2004, 09:31:13

Copyright (C) 2000-2004 CollabNet.
Subversion is open source software, see http://subversion.tigris.org/
This product includes software developed by CollabNet (http://www.Collab.Net/).

The following repository access (RA) modules are available:

* ra_dav : Module for accessing a repository via WebDAV (DeltaV) protocol.
  - handles 'http' schema
  - handles 'https' schema
* ra_local : Module for accessing a repository on local disk.
  - handles 'file' schema
* ra_svn : Module for accessing a repository using the svn network protocol.
  - handles 'svn' schema
```

RA-DAV (Acceso al repositorio usando HTTP/DAV)

La librería libsvn_ra_dav está diseñada para ser usada por clientes que están siendo ejecutado en diferentes máquinas que los servidores con los que se comunican, específicamente servidores a los que se ha alcanzado usando URLs que contienen las porciones de protocolo `http:` o `https:`. Para entender cómo funciona este módulo, deberíamos mencionar primero un grupo de componentes clave en esta configuración particular de capa de acceso al repositorio—el poderoso servidor HTTP Apache, y la librería cliente Neon HTTP/WebDAV.

El servidor principal de red de Subversion es el servidor HTTP Apache. Apache es un proceso servidor open-source extensible probado por mucho tiempo, preparado para uso serio. Puede soportar una elevada carga de red y se ejecuta en muchas plataformas. El servidor Apache soporta una variedad de protocolos de autenticación estándar, y puede expandirse por medio de módulos para soportar muchos más. También soporta optimizaciones de red como pipelining y caching. Usando Apache como servidor, Subversion obtiene todas estas características gratuitamente. Y dado que la mayoría de los cortafuegos ya permiten que pase el tráfico HTTP, los administradores de sistemas normalmente no tienen que cambiar siquiera la configuración de su cortafuegos para permitir que Subversion funcione.

Subversion usa HTTP y WebDAV (junto con DeltaV) para comunicarse con un servidor Apache. Puede leer más sobre esto en la sección WebDAV de este capítulo, pero en pocas palabras, WebDAV y DeltaV son extensiones al protocolo HTTP 1.1 estándar que permiten compartir y versionar ficheros a través de la web. Apache 2.0 viene con mod_dav, un módulo Apache que entiende las extensiones DAV de HTTP. El propio Subversion proporciona mod_dav_sv, que es otro módulo de Apache que funciona junto con mod_dav (realmente, es su motor) para proporcionar implementaciones específicas de Subversion de WebDAV y DeltaV.

Cuando se comunica con un repositorio por HTTP, la librería cargadora RA selecciona libsvn_ra_dav como el módulo correcto de acceso. El cliente de Subversion realiza llamadas sobre la interfaz genérica RA, y libsvn_ra_dav relaciona esas llamadas (que tienden a englobar acciones Subversion a gran escala) con un conjunto de peticiones HTTP/WebDAV. Usando la librería Neon, libsvn_ra_dav transmite esas peticiones al servidor Apache. Apache recibe estas peticiones (exactamente igual que con las peticiones genéricas HTTP que su navegador pudiera hacer), se da cuenta de que estas peticiones están dirigidas a una URL que está configurada como una ubicación DAV (usando la directiva `Location` en `httpd.conf`), y pasa la petición a su propio módulo mod_dav. Configurado correctamente, mod_dav sabe que debe usar mod_dav_svn de Subversion para cualquier necesidad relacionada con el sistema de ficheros, a diferencia del módulo genérico mod_dav_fs que viene con Apache. Así que finalmente el cliente se comunica con mod_dav_svn, que enlaza directamente con la capa de repositorio de Subversion.

Pero eso es una descripción simplificada de los intercambios reales que ocurren. Por ejemplo, el repositorio Subversion podría estar protegido por las directivas de autorización de Apache. Esto podría significar que los intentos iniciales de comunicación con el

repositorio podrían ser rechazados por Apache debido al código de autorización. En esta situación, libsvn_ra_dav recibe la notificación de Apache de que no se proporcionaron suficientes datos de identificación, y llama de vuelta a la capa cliente para obtener algunos datos actualizados de autenticación. Si los datos se proporcionan correctamente, y el usuario tiene los permisos que Apache necesita, el siguiente intento automático de libsvn_ra_dav para realizar la operación original será aceptado, y todo irá bien. Si no se puede proporcionar suficiente información de autenticación, la petición fallará, y el cliente informará del fallo al usuario.

Gracias al uso de Neon y Apache, Subversion también obtiene funcionalidad gratuita en algunas otras áreas complejas. Por ejemplo, si Neon encuentra las librerías OpenSSL, permitirá al cliente de Subversion intentar usar comunicaciones cifradas con SSL con el servidor Apache (cuyo módulo mod_ssl puede "hablar el lenguaje"). Además, tanto Neon como mod_deflate de Apache pueden entender el algoritmo "deflate" (el mismo usado por los programas PKZIP y gzip), por lo que las peticiones pueden ser enviadas en trozos comprimidos más pequeños a través del cable. Otras características complejas que Subversion espera soportar en el futuro incluyen la capacidad de manejar de forma automática las redirecciones indicadas por el servidor (por ejemplo, cuando un repositorio se mueve a otra nueva URL canónica) y obtener la ventaja del pipelining HTTP.

RA-SVN (Acceso al repositorio usando protocolo propio)

Además del protocolo estándar HTTP/WebDAV, Subversion también proporciona una implementación RA que usa un protocolo propio. El módulo libsvn_ra_svn implementa su propia conectividad por red, y se comunica con un servidor autosuficiente—el programa `svnserve`— en la máquina que almacena el repositorio. Los clientes acceden al repositorio usando el esquema `svn://`.

Esta implementación RA carece de la mayoría de las ventajas de Apache mencionadas en la sección anterior; no obstante, puede ser atractiva para algunos administradores de sistemas. Es dramáticamente más fácil de configurar y ejecutar; configurar un proceso `svnserve` es casi instantáneo. También es mucho más pequeño (en líneas de código) que Apache, haciéndolo más fácil de auditar, por razones de seguridad u otras. Además, algunos administradores de sistemas pueden tener ya instalada una infraestructura de seguridad SSH, que quieren que Subversion aproveche. Los clientes usando ra_svn pueden canalizar fácilmente el protocolo por SSH.

RA-Local (Acceso directo al repositorio)

No todas las comunicaciones con un repositorio Subversion requieren un todopoderoso proceso servidor y una capa de red. Para los usuarios que simplemente desean acceder a los repositorios de sus discos duros locales, pueden hacerlo usando URLs `file:` y la funcionalidad proporcionada por libsvn_ra_local. Este módulo RA enlaza directamente con las librerías de repositorio y sistema de ficheros, por lo que no se requiere comunicación por red en absoluto.

Subversion requiere que el nombre del servidor incluido como parte de la URL `file:` esté vacío o sea `localhost`, y que no exista especificación alguna de puerto. En otras palabras, las URLs deberían ser como `file://localhost/ruta/al/repositorio` o `file:///ruta/al/repositorio`.

Además, tenga en cuenta que las URLS `file:` de Subversion no pueden ser usadas en un navegador normal al igual que con la URL `file:` típica. Cuando intenta visualizar una URL `file:` en un navegador normal, lee y muestra el contenido del fichero que está en ese lugar examinando el sistema de ficheros directamente. No obstante, los recursos de Subversion existen en un sistema de ficheros virtual (vea "Capa de repositorio"), y su navegador no será capaz de entender cómo leer este sistema de ficheros.

Su librería RA aquí

Para aquellos que deseen acceder al repositorio Subversion usando todavía otro protocolo, ¡para eso precisamente está modularizada la capa de acceso al repositorio! Los desarrolladores pueden simplemente escribir una nueva librería que implemente la interfaz RA en un lado y se comunique con el repositorio por el otro. Su nueva librería puede usar protocolos de red existentes, o puede inventarse uno propio. Puede usar llamadas de comunicación inter-proceso (IPC), o—tiremos la casa por la ventana, ¿de acuerdo—podría incluso implementar un protocolo basado en email. Subversion proporciona las APIs; usted proporciona la creatividad.

Capa cliente

En el lado del cliente, la copia local de trabajo de Subversion es donde ocurre toda la acción. El grueso de la funcionalidad implementada por las librerías en el lado del cliente existe con el único propósito de gestionar copias locales de trabajo—directorios lle-

nos de ficheros y otros subdirectorios que sirven a modo de “reflejo” local y editable de una o más ubicaciones de repositorio—y propagar cambios hacia y desde la capa de acceso al repositorio.

La librería de Subversion de copias locales de trabajo, libsvn_wc, es directamente responsable de gestionar los datos en las copias locales. Para realizar esta tarea, la librería almacena información administrativa sobre cada directorio dentro de un subdirectorio especial. Este subdirectorio, llamado `.svn`, está presente en cada directorio de la copia local y contiene varios ficheros y directorios que almacenan el estado y proporcionan un espacio de trabajo privado para acciones administrativas. Para aquellos familiarizados con CVS, este subdirectorio `.svn` es similar en propósito a los directorios administrativos `CVS` encontrados en las copias de trabajo locales de CVS. Para más información sobre el área administrativa `.svn`, vea “Dentro del área de administración de la copia local de trabajo”en este capítulo.

La librería cliente de Subversion, libsvn_client, tiene la más amplia responsabilidad; su trabajo es reunir la funcionalidad de la librería de copia local de trabajo y la de la capa de acceso al repositorio, y entonces proporcionar la API de mayor nivel para cualquier aplicación que desee realizar acciones generales de control de revisiones. Por ejemplo, la función `svn_client_checkout` recibe una URL como parámetro. La función pasa la URL a la capa RA y abre una sesión autenticada con un repositorio en particular. Entonces le pregunta al repositorio por un árbol concreto y envía este árbol a la librería de copia local de trabajo, la cual escribe entonces una copia local de trabajo completa en el disco (directorios `.svn` y demás incluidos).

La librería cliente está diseñada para que pueda ser usada por cualquier aplicación. Aunque el código fuente de Subversion incluye un cliente estándar de línea de comando, debería ser muy sencillo escribir cualquier número de clientes gráficos sobre esta librería cliente. Nuevas interfaces gráficas (o realmente, cualquier cliente) para Subversion no tienen que ser envoltorios precarios sobre el cliente incluido de línea de comando—tienen acceso completo vía la API de libsvn_client a la misma funcionalidad, datos, y mecanismos de retrollamada usados por el cliente de línea de comando.

Acceso directo—unas palabras sobre corrección

¿Por qué debería su programa de interfaz gráfica acceder directamente a libsvn_client en lugar de actuar como envoltorio sobre el programa de línea de comando? Aparte de ser simplemente más eficiente, esto puede solucionar potenciales problemas de corrección. Un programa de línea de comando (como el que trae Subversion) que accede a la librería cliente necesita traducir con efectividad las respuestas y bits de datos solicitados desde tipos de datos de C a alguna forma de salida legible por un ser humano. Este tipo de traducción puede tener pérdidas. Es decir, un programa puede que no muestre toda la información cosechada a través de la API, o puede combinar bits de información para compactar la representación.

Si envuelve a un programa de línea de comando de este tipo con otro programa, el segundo sólo tiene acceso a la información ya interpretada (que como hemos mencionado, es posiblemente incompleta), que *de nuevo* debe ser traducida a *su* formato de representación. Con cada capa de envoltorio, la integridad de los datos originales es potencialmente corrompida más y más, igual que el resultado de hacer una copia de una copia (de una copia …) de su cinta de audio o vídeo favorita.

Usando las APIs

Desarrollar aplicaciones usando las APIs de las librerías de Subversion es bastante sencillo. Todos los ficheros de cabecera públicos se encuentran en el directorio `subversion/include` del código fuente. Estos ficheros de cabecera son copiados en su sistema cuando compila e instala Subversion a partir del código fuente. Estos ficheros de cabecera recogen por completo las funciones y tipos de datos accesibles por los usuarios de las librerías de Subversion.

La primera cosa que puede observar es que los tipos de datos y funciones de Subversion están protegidos por un espacio de nombrado . Todo símbolo público comienza con `svn_`, seguido por un breve código que representa la librería en la cual es definido (como `wc`, `client`, `fs`, etc.), seguido por un único carácter de subrayado (_) y por último el resto del nombre del símbolo. Las funciones semi públicas (usadas por varios ficheros de código fuente de una librería pero no fuera de ésta, y contenidas en los propios directorios de las librerías) difieren de este esquema de nombres en el uso del carácter de subrayado tras el código de librería, ya que usan doble carácter de subrayado (__). Las funciones que son privadas de un fichero particular no tienen prefijos especiales, y son declaradas como `static`. Por supuesto, el compilador no tiene interés alguno por estas convenciones de nombrado, pero ayudan a clarificar el ámbito de una función o tipo de datos.

La librería Apache Portable Runtime

Junto con los tipos de datos de Subversion, verá muchas referencias a tipos de datos que comienzan con `apr_`—símbolos de la librería Apache Portable Runtime (APR). APR es la librería de portabilidad de Apache, originada a partir del código fuente del servidor en un intento de separar el código dependiente del SO de las porciones de código independientes del SO. El resultado es una librería que proporciona una API genérica para realizar operaciones que difieren ligeramente —o terriblemente— de SO a SO. Aunque el servidor HTTP Apache fue obviamente el primer usuario de la librería APR, los desarrolladores de Subversion reconocieron inmediatamente el valor de usar también APR. Esto significa que prácticamente no hay porciones de código dependiente del SO en Subversion. También, significa que el cliente Subversion se puede compilar y ejecutar dondequiera que lo haga el servidor. Actualmente esta lista incluye todas las variantes de Unix, Win32, BeOS, OS/2, y Mac OS X.

Además de proporcionar implementaciones consistentes de llamadas de sistema que difieren entre sistemas operativos, [4] APR le da a Subversion acceso inmediato a muchos tipos de datos, como arrays dinámicos o tablas hash. Subversion usa en gran medida estos tipos a lo largo del código fuente. Pero quizás el tipo de datos APR más usado, encontrado en casi todo prototipo de la API de Subversion, es `apr_pool_t`—el área de memoria de APR. Subversion usa internamente áreas de memoria para todas las peticiones de reserva de memoria que necesita (excepto cuando una librería externa requiere un esquema de gestión de memoria diferente para los datos que pasan por su API), [5] y aunque una persona que programa usando las APIs de Subversion no está obligada a hacer lo mismo, tiene que proporcionar áreas de memoria a las funciones de la API que los necesitan. Esto significa que los usuarios de la API de Subversion también tienen que enlazar con APR, y deben llamar `apr_initialize()` para iniciar el subsistema APR, y entonces adquirir un área de memoria para usarla con las llamadas de la API de Subversion. Vea “Programando con áreas de memoria” para más información.

Requisitos de URL y ruta

Con operaciones de control de versiones remotas como razón de la existencia de Subversion, tiene sentido prestar atención al soporte de internacionalización (i18n[6]). Después de todo, aunque “remotas” pueda significar “de un lado a otro de la oficina”, podría significar perfectamente “de un lado a otro del globo.”. Para facilitar esto, todas las interfaces públicas de Subversion que aceptan parámetros de rutas esperan que éstas sean canónicas, y codificadas en UTF-8. Esto significa, por ejemplo, que cualquier nuevo cliente binario que use la interfaz libsvn_client necesita convertir primero las rutas de la codificación específica local a UTF-8 antes de pasarlas a las librerías de Subversion, y entonces reconvertir cualquier resultado generado por Subversion de nuevo en la codificación local antes de usar esas rutas con propósitos no relacionados con Subversion. Afortunadamente, Subversion proporciona un conjunto de funciones (vea `subversion/include/svn_utf.h`) que pueden ser usadas por cualquier programa para realizar estas conversiones.

Además, las APIs de Subversion requieren que todos los parámetros con URLs estén codificados correctamente como URIs. Así, que en lugar de pasar `file:///home/nombreusuario/Mi fichero.txt` como la URL de un fichero llamado `Mi fichero.txt`, usted necesita pasar `file:///home/nombreusuario/Mi%20fichero.txt`. De nuevo, Subversion proporciona funciones de ayuda que su aplicación puede usar—`svn_path_uri_encode` y `svn_path_uri_decode`, para codificar y decodificar URIs, respectivamente.

Usando lenguajes distintos de C y C++

Si está interesado en usar las librerías de Subversion junto con algo que no sea un programa en C—digamos que un script en Python o una aplicación en Java—Subversion tiene un soporte básico para esto vía el generador simplificado de envoltorios e interfaces (SWIG) [7]. Los enlaces SWIG para Subversion se encuentran en `subversion/bindings/swig` y están madurando lentamente hasta llegar a un estado usable. Estos enlaces le permiten realizar llamadas a funciones de la API de Subversion indirectamente, usando los envoltorios que traducen los tipos de datos nativos de su lenguaje de script a los tipos de datos necesarios por las librerías C de Subversion.

Hay otro beneficio al acceder a las APIs de Subversion a través de un enlace con otro lenguaje—simplicidad. En general, lenguajes como Python y Perl son mucho más flexibles y fáciles de usar que C o C++. Los tipos de datos de alto nivel y la verificación de tipos por contexto proporcionados por estos lenguajes son a menudo mejores manejando la información que viene de los usuarios.

[4] Subversion usa llamadas de sistema y tipos de datos ANSI siempre que es posible.
[5] Neon y la base de datos de Berkeley son tales librerías.
[6] N.T.: Forma breve de referirse a la palabra internacionalización, en inglés “internationalization”, que comienza con una “i”, seguida de 18 caracteres y terminada en “n”.
[7] N.T.: “Simplified wrapper and interface generator” en inglés

Como ya sabrá, los humanos son muy hábiles fastidiando los datos de entrada de un programa, y los lenguajes de script tienden a manejar tales incorrecciones con más gracilidad. Por eso usar una interfaz y conjunto de librerías altamente optimizados, basados en C, combinados con un lenguaje poderoso, flexible, es tan atractivo.

Echemos un vistazo a un ejemplo que usa los envoltorios SWIG de Subversion para Python. Nuestro ejemplo hará exactamente lo mismo que el último. ¡Fíjese en la diferencia de tamaño y complejidad de esta versión de la función!

Ejemplo 8.2. Usando la capa de repositorio con Python

```
from svn import fs
import os.path

def crawl_filesystem_dir (root, directory, pool):
  """Recursively crawl DIRECTORY under ROOT in the filesystem, and return
  a list of all the paths at or below DIRECTORY.  Use POOL for all
  allocations."""

  # Get the directory entries for DIRECTORY.
  entries = fs.dir_entries(root, directory, pool)

  # Initialize our returned list with the directory path itself.
  paths = [directory]

  # Loop over the entries
  names = entries.keys()
  for name in names:
    # Calculate the entry's full path.
    full_path = os.path.join(basepath, name)

    # If the entry is a directory, recurse.  The recursion will return
    # a list with the entry and all its children, which we will add to
    # our running list of paths.
    if fs.is_dir(fsroot, full_path, pool):
      subpaths = crawl_filesystem_dir(root, full_path, pool)
      paths.extend(subpaths)

    # Else, it is a file, so add the entry's full path to the FILES list.
    else:
      paths.append(full_path)

  return paths
```

Una implementación en C del ejemplo anterior se alargaría bastante más. La misma rutina en C necesitaría prestar especial atención al uso de la memoria, y tendría que usar tipos de datos de propios para representar las tablas hash y la lista de rutas. Python tiene tablas hash (llamadas “diccionarios”) y listas como tipos de datos nativos, y proporciona una maravillosa selección de métodos para operar sobre esos tipos de datos. Y dado que Python usa recolección de basura y recuento de referencias, los usuarios del lenguaje no necesitan preocuparse con la reserva y liberación de memoria.

En la sección previa de este capítulo, mencionamos la interfaz `libsvn_client`, y cómo existe con el único propósito de simplificar el proceso de escribir un cliente de Subversion. A continuación presentamos un breve ejemplo que muestra cómo esta librería puede ser usada vía los envoltorios SWIG. ¡En sólo unas pocas línea de Python, puede obtener una copia de trabajo local de Subversion totalmente funcional!

Ejemplo 8.3. Un script simple para obtener una copia de trabajo local.

```
#!/usr/bin/env python
import sys
from svn import util, _util, _client

def usage():
  print "Usage: " + sys.argv[0] + " URL PATH\n"
  sys.exit(0)

def run(url, path):
  # Initialize APR and get a POOL.
  _util.apr_initialize()
  pool = util.svn_pool_create(None)

  # Checkout the HEAD of URL into PATH (silently)
  _client.svn_client_checkout(None, None, url, path, -1, 1, None, pool)

  # Cleanup our POOL, and shut down APR.
  util.svn_pool_destroy(pool)
  _util.apr_terminate()

if __name__ == '__main__':
  if len(sys.argv) != 3:
    usage()
  run(sys.argv[1], sys.argv[2])
```

Los envoltorios de Subversion para otros lenguajes desafortunadamente tienden a no recibir el mismo nivel de atención recibido por los módulos principales de Subversion. No obstante, se han dado pasos significativos hacia la creación de enlaces funcionales para Python, Perl y Java. Una vez tenga los ficheros del interfaz SWIG configurados correctamente, la generación de envoltorios específicos para todos los lenguajes soportados por SWIG (que actualmente incluye versiones de C#, Guile, Java, MzScheme, OCaml, Perl, PHP, Python, Ruby y Tcl) debería ser teóricamente trivial. A pesar de esto, todavía se requiere un poco de programación extra para compensar las complejas APIs con las que SWIG necesita ayuda. Para más información sobre SWIG, vea la página web del proyecto en `http://www.swig.org/`.

Dentro del área de administración de la copia local de trabajo

Tal y como mencionamos anteriormente, cada directorio de una copia local de trabajo realizada con Subversion contiene un subdirectorio especial llamado `.svn` que almacena datos administrativos sobre ese directorio local de trabajo. Subversion usa la información de `.svn` para llevar la pista de cosas como:

- Qué lugar(es) del repositorio está representado por los ficheros y subdirectorios del directorio de la copia local de trabajo.
- Qué revisión de cada uno de esos ficheros y directorios está actualmente presente en la copia local.
- Cualquier atributo definido por el usuario que pudiese estar asociado a esos ficheros y directorios.
- Copias prístinas (no editadas) de los ficheros de la copia local de trabajo.

A pesar de que hay varios otros datos almacenados en el directorio `.svn`, examinaremos sólo un puñado de los elementos más importantes.

El fichero de entradas

Quizás el fichero más importante en el directorio `.svn` es el fichero `entries`. El fichero de entradas es un documento XML que contiene el grueso de la información administrativa sobre un recurso versionado en el directorio de la copia local de trabajo. Es el fichero que lleva la pista a las URLs de repositorio, revisiones prístinas, sumas de control de ficheros, textos prístinos y fechas de modificación de propiedades, información de planificación y estado de conflicto, última información conocida de cambios en el repositorio (autor, revisión, fecha de modificación), historia de la copia local—¡prácticamente todo lo que un cliente de Subversion puede estar interesado en saber sobre un recurso versionado (o por versionar)!

Comparando las áreas administrativas de Subversion y CVS

Un vistazo dentro del típico directorio `.svn` revela más cosas que lo que CVS mantiene en su directorio administrativo `CVS`. El fichero `entries` contiene XML que describe el estado actual del directorio de la copia local de trabajo, y básicamente sirve para los propósitos de los ficheros de CVS `Entries`, `Root`, y `Repository` combinados.

A continuación mostramos un ejemplo de un fichero de entradas real:

Ejemplo 8.4. Contenido de un fichero `.svn/entries` típico.

```
<?xml version="1.0" encoding="utf-8"?>
<wc-entries
   xmlns="svn:">
<entry
   committed-rev="1"
   name="svn:this_dir"
   committed-date="2002-09-24T17:12:44.064475Z"
   url="http://svn.red-bean.com/tests/.greek-repo/A/D"
   kind="dir"
   revision="1"/>
<entry
   committed-rev="1"
   name="gamma"
   text-time="2002-09-26T21:09:02.000000Z"
   committed-date="2002-09-24T17:12:44.064475Z"
   checksum="QSE4vWd9ZM0cMvr7/+YkXQ=="
   kind="file"
   prop-time="2002-09-26T21:09:02.000000Z"/>
<entry
   name="zeta"
   kind="file"
   schedule="add"
   revision="0"/>
<entry
   url="http://svn.red-bean.com/tests/.greek-repo/A/B/delta"
   name="delta"
   kind="file"
   schedule="add"
   revision="0"/>
<entry
   name="G"
   kind="dir"/>
<entry
   name="H"
   kind="dir"
   schedule="delete"/>
</wc-entries>
```

Tal y como puede ver, el fichero entries es esencialmente una lista de entradas. Cada etiqueta `entry` representa una de tres posibles cosas: el propio directorio de la copia local (entrada conocida como “este directorio”, y caracterizada por tener un valor vacío para su atributo *`name`*), un fichero en esa copia local (caracterizado por tener en su atributo *`kind`* el valor `"file"`), o un subdirectorio en esa copia local (*`kind`* en este caso tendrá el valor `"dir"`). Los ficheros y subdirectorios cuyas entradas ya están almacenadas en este fichero están o bien bajo control de versiones, o (como en el caso del fichero `zeta` más arriba) están programados para ser puestos bajo control de versiones la siguiente vez que el usuario envíe los cambios de este directorio al repositorio. Cada entrada tiene un nombre único, y cada entrada es de tipo nodo .

Los desarrolladores deben tener presentes algunas reglas especiales que Subversion usa cuando lee y escribe sus ficheros `entries`. Aunque cada entrada tiene asociada una revisión y URL, fíjese que no toda etiqueta `entry` en el fichero de ejemplo tiene un atributo *`revision`* o *`url`* asociado a ella. Subversion permite que las entradas no almacenen explícitamente esos dos atributos cuando sus valores son iguales que (en el caso de *`revision`*) o triviales de calcular a partir de [8] (en el caso de *`url`*) los datos almacenados en la entrada “este directorio”. Tenga también en cuenta que para las entradas de subdirectorios, Subversion almacena únicamente los atributos cruciales—nombre, tipo, url, revisión, y acción programada. En un esfuerzo para reducir información duplicada, Subversion dicta que el método para determinar el conjunto completo de información sobre un subdirectorio es descender dentro de ese subdirectorio, y leer la entrada “este directorio” de su propio fichero `.svn/entries`. No obstante, se mantiene una referencia al subdirectorio en el fichero `entries` del padre, con suficiente información para permitir operaciones básicas de versionado en el evento de que el propio subdirectorio no esté presente en el disco.

Copias prístinas y propiedades de ficheros

Tal y como mencionamos anteriormente, el directorio `.svn` también contiene versiones “text-base” prístinas de los ficheros. Éstas pueden encontrarse en `.svn/text-base`. Los beneficios de estas copias son múltiples—evitan conexiones por red cuando se quiere saber si hay modificaciones locales y qué es lo que ha cambiado, permiten revertir ficheros modificados o eliminados, los cambios que se transmiten al servidor son menores—pero hay que pagar el precio de almacenar cada fichero versionado dos veces en el disco. Hoy en día, esto parece una desventaja despreciable para la mayoría de los ficheros. No obstante, esta situación se vuelve más delicada a medida que el tamaño de sus ficheros versionados crece. Se está prestando atención para permitir que la presencia de “text-base” sea una opción. Pero irónicamente, es a medida que los tamaños de sus ficheros versionados crecen cuando la existencia de “text-base” se vuelve más crucial—¿quién quiere transmitir un fichero enorme por red sólo porque quieren realizar una pequeña modificación sobre el mismo?

Con un propósito similar a los ficheros “text-base” existen los ficheros de propiedades y sus copias “prop-base” prístinas, encontradas en `.svn/props` y `.svn/prop-base` respectivamente. Dado que los directorios también pueden tener propiedades, también existen los ficheros `.svn/dir-props` y `.svn/dir-prop-base`. Cada uno de estos ficheros de propiedades (versiones “base” y “en desarrollo”) usan un simple fichero de formato “hash-on-disk” para almacenar los nombres de las propiedades y sus valores.

WebDAV

WebDAV (acrónimo de “Web-based Distributed Authoring and Versioning”[9]) es una extensión del protocolo estándar HTTP diseñada para transformar la web en un medio de lectura y escritura, en lugar del medio básicamente de sólo lectura que existe hoy en día. La teoría es que los directorios y ficheros pueden ser compartidos—tanto como objetos de lectura y escritura—sobre la web. Los RFCs 2518 y 3253 describen las extensiones WebDAV/DeltaV de HTTP, y están disponibles (junto con bastante información útil) en `http://www.webdav.org/`.

Varios navegadores de ficheros en diferentes sistemas operativos ya son capaces de montar directorios por red usando WebDAV. En Win32, el explorador de Windows puede navegar por lo que llama WebFolders (que son ubicaciones red montadas con WebDAV) como si fuesen normales carpetas compartidas. Mac OS X también tiene esta capacidad, al igual que los navegadores Nautilus y Konqueror (bajo GNOME y KDE, respectivamente).

¿Cómo afecta todo esto a Subversion? El módulo mod_dav_svn de Apache usa HTTP, extendido por WebDAV y DeltaV, como uno de sus protocolos de red. Subversion usa mod_dav_svn para relacionar los conceptos de versionado de Subversion con los de

[8] Es decir, la URL de la entrada es igual que la concatenación de la URL del directorio padre y el nombre de la entrada.
[9] N.T.: Gestión y versionado distribuido vía web

los RFCs 2518 y 3253.

Puede encontrar una descripción más detallada de WebDAV, cómo funciona, y cómo es usado por Subversion en Apéndice C, *WebDAV y autoversionado*. Entre otras cosas, ese apéndice discute el grado con el que Subversion se adhiere a la especificación WebDAV genérica, y cómo eso afecta la interoperabilidad con clientes WebDAV genéricos.

Programando con áreas de memoria

Casi todo desarrollador que ha usado el lenguaje de programación C se ha exasperado ante la desalentadora tarea de gestionar la memoria. Reservar suficiente memoria para un uso, llevar la pista de estas reservas, liberar la memoria cuando ya no se necesita—estas tareas pueden ser bastante complejas. Y por supuesto, errar en estas operaciones puede llevar a un programa que se cuelga, o peor, que cuelga todo el ordenador. Afortunadamente, la librería APR, de la que depende Subversion para su portabilidad, proporciona el tipo de dato apr_pool_t, el cual representa un área de la cual la aplicación puede reservar memoria.

Un área de memoria es una representación abstracta de un bloque de memoria reservado para ser usado por un programa. En lugar de pedir memoria directamente al SO usando las funciones estándar `malloc()` y compañía, los programas que enlazan con APR pueden simplemente solicitar que se cree un área de memoria (usando la función `apr_pool_create()`). APR reservará un trozo de memoria de tamaño moderado del SO, y esa memoria será disponible instantáneamente para ser usada por el programa. En cualquier momento que el programa necesite algo de memoria, usará una de las funciones de área de memoria de la API APR, como `apr_palloc()`, la cual devuelve una dirección genérica de memoria que apunta dentro del área. El programa puede seguir pidiendo bits y trozos de memoria del área, y APR seguirá accediendo a las demandas. Las áreas de memoria crecen automáticamente en tamaño para acomodarse a los programas que solicitan más memoria que la contenida inicialmente por el área, hasta que por supuesto, no quede más memoria disponible en el sistema.

Ahora bien, si este fuese el final de la historia del área de memoria, difícilmente se hubiese merecido atención especial. Afortunadamente este no es el caso. Las áreas no sólo pueden ser creadas; también pueden ser limpiadas y destruidas, usando `apr_pool_clear()` y `apr_pool_destroy()` respectivamente. Esto proporciona a los desarrolladores la flexibilidad de reservar varios—o varios miles—elementos del área, ¡y entonces limpiar toda esa memoria con una simple llamada! Además, las áreas de memoria tienen jerarquía. Puede hacer “sub áreas” de cualquier área de memoria previamente creada. Cuando limpia un área, todas sus sub áreas son destruidas; si destruye un área, ésta y sus sub áreas serán destruidas.

Antes de continuar, los desarrolladores deben tener presente que probablemente no encontrarán muchas llamadas a las funciones de áreas de memoria APR que acabamos de mencionar en el código fuente de Subversion. Las áreas de memoria APR ofrecen algunos mecanismos de extensión, como la posibilidad de tener “datos de usuario” personalizados asociados al área de memoria, y mecanismos para registrar funciones de limpieza que serán llamadas cuando el área de memoria sea destruida. Subversion hace uso de estas extensiones de un modo no trivial. Por lo que Subversion proporciona (y la mayoría del código usa) las funciones de envoltorio `svn_pool_create()`, `svn_pool_clear()`, y `svn_pool_destroy()`.

Aunque las áreas de memoria son útiles para tareas básicas de gestión de memoria, este tipo de estructura realmente brilla con los bucles y la recursividad. Dado que los bucles a menudo pueden terminar en cualquier momento , y las funciones recursivas volver a cualquier profundidad, el consumo de memoria en este tipo de situaciones puede ser impredecible. Afortunadamente, usar áreas de memoria anidadas puede ser un método excelente para tratar con estas situaciones peliagudas. El siguiente ejemplo demuestra el uso básico de áreas de memoria anidadas en una situación que es bastante común—recorrer de forma recursiva un árbol de directorios realizando cierta tarea sobre cada elemento del árbol.

Ejemplo 8.5. Uso efectivo de áreas de memoria

```
/* Recursively crawl over DIRECTORY, adding the paths of all its file
   children to the FILES array, and doing some task to each path
   encountered.  Use POOL for the all temporary allocations, and store
   the hash paths in the same pool as the hash itself is allocated in.  */
static apr_status_t
crawl_dir (apr_array_header_t *files,
           const char *directory,
           apr_pool_t *pool)
```

```
{
  apr_pool_t *hash_pool = files->pool;  /* array pool */
  apr_pool_t *subpool = svn_pool_create (pool);  /* iteration pool */
  apr_dir_t *dir;
  apr_finfo_t finfo;
  apr_status_t apr_err;
  apr_int32_t flags = APR_FINFO_TYPE | APR_FINFO_NAME;

  apr_err = apr_dir_open (&dir, directory, pool);
  if (apr_err)
    return apr_err;

  /* Loop over the directory entries, clearing the subpool at the top of
     each iteration.  */
  for (apr_err = apr_dir_read (&finfo, flags, dir);
       apr_err == APR_SUCCESS;
       apr_err = apr_dir_read (&finfo, flags, dir))
    {
      const char *child_path;

      /* Clear the per-iteration SUBPOOL.  */
      svn_pool_clear (subpool);

      /* Skip entries for "this dir" ('.') and its parent ('..').  */
      if (finfo.filetype == APR_DIR)
        {
          if (finfo.name[0] == '.'
              && (finfo.name[1] == '\0'
                  || (finfo.name[1] == '.' && finfo.name[2] == '\0')))
            continue;
        }

      /* Build CHILD_PATH from DIRECTORY and FINFO.name.  */
      child_path = svn_path_join (directory, finfo.name, subpool);

      /* Do some task to this encountered path. */
      do_some_task (child_path, subpool);

      /* Handle subdirectories by recursing into them, passing SUBPOOL
         as the pool for temporary allocations.  */
      if (finfo.filetype == APR_DIR)
        {
          apr_err = crawl_dir (files, child_path, subpool);
          if (apr_err)
            return apr_err;
        }

      /* Handle files by adding their paths to the FILES array.  */
      else if (finfo.filetype == APR_REG)
        {
          /* Copy the file's path into the FILES array's pool.  */
          child_path = apr_pstrdup (hash_pool, child_path);

          /* Add the path to the array.  */
          (*((const char **) apr_array_push (files))) = child_path;
        }
    }

  /* Destroy SUBPOOL.  */
  svn_pool_destroy (subpool);

  /* Check that the loop exited cleanly. */
  if (apr_err)
    return apr_err;
```

```
  /* Yes, it exited cleanly, so close the dir. */
  apr_err = apr_dir_close (dir);
  if (apr_err)
    return apr_err;

  return APR_SUCCESS;
}
```

El ejemplo anterior demuestra un uso efectivo de áreas de memoria en situaciones con bucles y recursividad. Cada recursión comienza creando una sub área de memoria del área pasada a la función. Esta sub área es usada por la región con bucles, y limpiada en cada iteración. Esto da como resultado que el consumo de memoria sea a grandes rasgos proporcional a la profundidad de la recursión, no al número total de ficheros y directorios presentes como hijos del directorio de nivel superior. Cuando la primera llamada a esta función recursiva finalmente termina, hay realmente muy pocos datos almacenados en el área de memoria que recibió como parámetro. ¡Ahora imagine la complejidad adicional que estaría presente en esta función si tuviese que reservar y liberar cada pieza individual de datos usados!

Quizás las áreas de memoria no sean ideales para todas las aplicaciones, pero son extremadamente útiles en Subversion. Como desarrollador de Subversion, tendrá que acostumbrarse a las áreas de memoria y a su uso correcto. Los fallos relacionados con el uso de memoria y exceso de código pueden ser difíciles de diagnosticar y corregir se la API que sea, pero la estructura de área de memoria proporcionada por APR ha resultado ser una funcionalidad tremendamente conveniente y ahorradora de tiempo.

Contribuyendo a Subversion

La fuente oficial de información sobre el proyecto Subversion es, por supuesto, la página web del proyecto en `http://subversion.tigris.org/`. Ahí puede encontrar información sobre cómo obtener acceso al código fuente y participar en las listas de discusión. La comunidad de Subversion siempre da la bienvenida a nuevos miembros. Si está interesado en participar en esta comunidad contribuyendo cambios al código fuente, aquí tiene algunas pistas para ponerse en marcha.

Únase a la comunidad

El primer paso para participar en la comunidad es encontrar un modo de estar al día de los últimos eventos. Para hacer esto de la manera más efectiva, querrá apuntarse a la lista principal de discusión de desarrolladores (`<dev@subversion.tigris.org>`) y la lista de cambios enviados al servidor (`<svn@subversion.tigris.org>`). Siguiendo estas listas incluso desde lejos, tendrá acceso a importantes discusiones de diseño, será capaz de ver los cambios en el código de Subversion a medida que ocurren, y presenciar revisiones de otras personas de esos cambios y propuestos cambios. Estas listas de discusión basadas en email son el medio de comunicación principal para el desarrollo de Subversion. Vea la sección de listas de correo en la página web para ver otras listas relacionadas con Subversion en las que pueda estar interesado.

¿Pero cómo puede saber qué es lo que debe hacerse? Es bastante habitual para un programador tener la mejor de las intenciones de ayudar con el desarrollo, pero ser incapaz de encontrar un buen punto de partida. Después de todo, no todas las personas vienen a la comunidad habiendo decidido qué detalle en particular quieren modificar. Pero observando las listas de discusión de los desarrolladores, puede llegar a ver mencionados fallos existentes o peticiones relacionadas con algo que le interese particularmente. También, otro buen lugar para buscar tareas pendientes no asignadas es la base de datos de tareas en la página web de Subversion. Ahí encontrará una lista actualizada de fallos conocidos y peticiones de características. Si quiere comenzar con algo pequeño, busque tareas marcadas con “bite-sized”.

Obtenga el código fuente

Para editar código fuente, necesita tener código fuente. Esto significa que debe obtener una copia local del repositorio público de código fuente de Subversion. A pesar de lo simple que parezca esta tarea, puede ser algo complicada. Dado que el código fuente de Subversion se versiona con el propio Subversion, en realidad necesita “arrancar” obteniendo un cliente de Subversion funcional por otro método. Los métodos más habituales incluyen descargar la última distribución binaria (si es que hay una disponible para su plataforma), o descargar el último paquete de código fuente y compilar su propio cliente de Subversion. Si compila el código fuente, asegúrese de leer las instrucciones del fichero `INSTALL` en el directorio raíz del código fuente.

Una vez disponga de un cliente de Subversion funcional, puede obtener una copia local del código fuente de Subversion de `http://svn.collab.net/repos/svn/trunk/`: [10]

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subversion
A  HACKING
A  INSTALL
A  README
A  autogen.sh
A  build.conf
…
```

El comando anterior obtendrá la ultima y más reciente versión del código de Subversion en un subdirectorio llamado `subversion` en su directorio actual. Obviamente, puede ajustar ese último parámetro como desee. Ignorando cómo llame al directorio de la copia local, una vez esta operación haya terminado, dispondrá del código fuente de Subversion. Por supuesto, todavía tendrá que obtener las librerías de apoyo (apr, apr-util, etc.)—vea el fichero `INSTALL` en el directorio raíz de su copia local para más detalles.

Familiarícese con las reglas de la comunidad

Ahora que tiene una copia local con la última versión del código fuente de Subversion, seguramente querrá leer el fichero `HACKING` en el directorio raíz de esa copia local. El fichero `HACKING` contiene instrucciones generales para contribuir con Subversion, incluyendo cómo formatear correctamente su código fuente para mantener la consistencia con el resto del código existente, cómo describir sus cambios propuestos con un mensaje de cambios efectivo, cómo verificar sus cambios, y demás. Los privilegios para hacer cambios en el repositorio del código fuente de Subversion se ganan—un gobierno basado en la meritocracia. [11] El fichero `HACKING` es un recurso de valor incalculable cuando llega la hora de asegurarse que sus cambios propuestos se ganan las alabanzas que merecen sin ser rechazados por detalles técnicos.

Realizando y verificando sus cambios

Habiendo entendido las reglas del código fuente y la comunidad, está preparado para realizar sus cambios. Lo mejor es intentar hacer cambios pequeños pero relacionados, incluso separando tareas grandes en fases, en lugar de hacer enormes modificaciones demoledoras. Los cambios que proponga serán más fáciles de entender (y por lo tanto, revisar) si distribuye el menor número de lineas de código posible para realizar la tarea correctamente. Tras realizar cada conjunto de cambios propuestos, su árbol Subversion debe estar en un estado en el cual el software compila sin ningún mensaje de aviso.

Subversion cuenta con una suite de tests de regresión bastante detallados [12], y se espera que los cambios que propone no harán que alguno de esos tests falle. Puede verificar sus cambios ejecutando **make check** (bajo Unix) desde el directorio raíz del código fuente. La forma más rápida para conseguir que sus contribuciones de código sean rechazadas (aparte de no proporcionar un buen mensaje de cambios) es proponer cambios que provocan fallos en la suite de tests.

En el mejor de los casos, usted habrá añadido nuevos tests a la suite de tests, los cuales verifican que sus cambios propuestos funcionan como espera. De hecho, a veces la mejor contribución que puede hacer una persona es añadir nuevos tests. Puede escribir tests de regresión para funcionalidades que actualmente funcionan en Subversion a modo de protección contra cambios futuros que puedan provocar un fallo en esas áreas. Además, puede escribir nuevos tests que demuestran fallos conocidos. Para este propósito, la suite de tests de Subversion le permite especificar que un test específico tiene como objetivo fallar (llamado `XFAIL`), así que mientras Subversion falle del modo que usted espera, el resultado del test `XFAIL` es considerado un éxito. En última instancia, cuanto mejor sea la suite de tests, menos tiempo se perderá diagnosticando potenciales fallos de regresión esquivos.

Donar sus cambios

[10] Tenga en cuenta que la URL obtenida en el ejemplo no finaliza con un `svn`, si no con uno de sus subdirectorios llamado `trunk`. Vea nuestra discusión sobre el modelo de ramas y etiquetado de Subversion para entender el razonamiento subyacente.

[11] Aunque esto pueda parecer superficialmente como una forma de elitismo, esta noción de “ganarse su privilegio de escritura” es una cuestión de eficiencia—es calcular si cuesta más tiempo y esfuerzo revisar y aplicar los cambios de otra persona que pueden ser seguros y útiles, versus el coste potencial de deshacer cambios que son peligrosos.

[12] Quizás quiera coger palomitas. “Detallados”, en este caso, se traduce en aproximadamente treinta minutos de computación no interactiva.

Tras realizar sus modificaciones al código fuente, componga un mensaje de cambios claro y conciso que los describa y la razón de éstos. Entonces, envíe un email a la lista de desarrolladores con su mensaje de cambios y la salida del comando **svn diff** (realizado desde el directorio raíz de su copia local de Subversion). Si los miembros de la comunidad consideran aceptables sus cambios, alguien con privilegios de escritura (permiso para crear nuevas revisiones en el repositorio de código fuente de Subversion) añadirá sus cambios al árbol público de código fuente. Recuerde que el permiso para realizar cambios directamente en el repositorio se obtiene por méritos—si demuestra comprensión de Subversion, competencia programando, y un “espíritu de equipo”, es probable que se le recompense con este permiso.

Capítulo 9. Referencia completa de Subversion

Este capítulo tiene el propósito de ser una referencia completa sobre el uso de Subversion. Incluye el cliente de línea de comando (**svn**) y todos sus subcomandos, al igual que los programas de administración del repositorio (**svnadmin** y **svnlook**) y sus respectivos subcomandos.

El cliente de línea de comando de Subversion: svn

Para usar el cliente de línea de comando, debe teclear **svn**, el subcomando que desea usar [1], y cualquier parámetro u objetivos sobre los cuales desea operar—no hay un orden específico en el cual deben aparecer los subcomandos y parámetros. Por ejemplo, todos los siguientes ejemplos de uso de **svn status** son válidos:

```
$ svn -v status
$ svn status -v
$ svn status -v myfile
```

Puede encontrar más ejemplos de cómo usar la mayoría de los comandos del cliente en Capítulo 3, *Recorrido guiado* y comandos para manejar propiedades en "Propiedades".

Parámetros de svn

Aunque Subversion tiene diferentes parámetros para sus subcomandos, todos los parámetros son globales—es decir, se garantiza que cada parámetro significa la misma cosa sin importar el subcomando que use. Por ejemplo, `--verbose` (`-v`) siempre significa "salida abundante de mensajes", sin importar el subcomando con el que lo use.

`--auto-props`

Activa auto propiedades, reemplazando la directiva `enable-auto-props` del fichero `config`.

`--config-dir` *`DIR`*

Indica a Subversion que lea la información de configuración del directorio especificado en lugar del lugar por defecto (`.subversion` en el directorio home del usuario).

`--diff-cmd` *`CMD`*

Especifica el programa externo que desea usar para mostrar las diferencias entre ficheros. Cuando invoca **svn diff**, usa el motor diferencial interno de Subversion, el cual genera ficheros diferenciales en formato unificado por defecto. Si quiere usar un programa de diferenciación externo, use `--diff-cmd`. Puede pasar parámetros a su programa de diferenciación con el parámetro `--extensions` (más sobre esto adelante en esta sección).

`--diff3-cmd` *`CMD`*

Especifica el programa externo que desea usar para fusionar los ficheros.

`--dry-run`

Recorre todo el proceso interno de la ejecución del comando, pero en realidad no se realizan cambios—ya sea en disco o en el repositorio.

[1] Si, de acuerdo, no necesita un subcomando para usar el parámetro `--version`, pero llegaremos a esto en un minuto.

`--editor-cmd` *CMD*

Especifica el programa externo que desea usar para modificar mensajes de informes de cambios o valores de propiedades.

`--encoding` *ENC*

Indica a Subversion que el mensaje que escribe al enviar cambios al repositorio está en la codificación indicada. Por defecto se usa la configuración local nativa de su sistema operativo, y debería especificar la codificación si sus mensajes de cambios están en otra codificación.

`--extensions` (`-x`) *ARGS*

Especifica un parámetro o varios que Subversion debe pasar al comando de diferenciación externo cuando se muestran diferencias entre ficheros. Si desea pasar múltiples argumentos, debe entrecomillarlos todos (por ejemplo, **svn diff --diff-cmd /usr/bin/diff -x "-b -E"**). Este parámetro *sólo* puede ser usado si también usa el parámetro `--diff-cmd`.

`--file` (`-F`) *FILENAME*

Usa el contenido del fichero pasado como argumento de este parámetro para el subcomando especificado.

`--force`

Obliga la ejecución de un comando u operación particular. Hay algunas operaciones que Subversion evitará hacer durante un uso normal, pero puede pasar el parámetro de forzado para decirle a Subversion “Se lo que estoy haciendo y las posibles repercusiones de mis actos, así que déjame hacerlos”. Este parámetro es el equivalente programático de realizar algún trabajo eléctrico sin desconectar la corriente—si no sabe lo que está haciendo, es probable que consiga una desagradable electrocución.

`--force-log`

Obliga la aceptación de parámetros sospechosos usados con las opciones `--message` (`-m`) o `--file` (`-F`). Por defecto, Subversion producirá un error si los parámetros de estas opciones perecen ser objetivos del subcomando. Por ejemplo, si pasa la ruta de un fichero versionado a la opción `--file` (`-F`), Subversion asumirá que ha cometido un fallo, y que esa ruta era en realidad el objetivo de la operación, y simplemente se equivocó al indicar el fichero—no versionado—que contiene el registro de mensajes de cambios. Para confirmar su intención y anular este tipo de errores, pase la opción `--force-log` a comandos que aceptan mensajes de cambios.

`--help` (`-h` o `-?`)

Si se usa con uno o más subcomandos, muestra la ayuda de texto empotrada en el cliente para cada subcomando. Si se usa sin nada más, muestra la ayuda de texto genérica del cliente.

`--ignore-ancestry`

Indica a Subversion que ignore la ascendencia cuando calcule diferencias (sólo usará el contenido de las rutas).

`--incremental`

Muestra la salida en un formato que permite la concatenación.

`--message` (`-m`) *MESSAGE*

Indica que especificará el mensaje del informe de cambios en la línea de comando, siguiendo al parámetro. Por ejemplo:

```
$ svn commit -m "They don't make Sunday."
```

`--new` *ARG*

Usa *ARG* como el objetivo nuevo.

`--no-auth-cache`

Previene el cacheado de información de autenticación (ej: nombre de usuario y clave de acceso) en los directorios administrativos de Subversion.

`--no-auto-props`

Desactiva las propiedades automáticas, reemplazando la directiva `enable-auto-props` del fichero `config`.

`--no-diff-deleted`

Evita que Subversion muestre diferencias para ficheros borrados. El comportamiento por defecto cuando elimina un fichero es que **svn diff** imprima las mismas diferencias que vería si hubiese dejado el fichero pero eliminando su contenido.

`--no-ignore`

Muestra ficheros en el listado de estado que normalmente serían omitidos debido a que coinciden con un patrón de la propiedad `svn:ignore`. Vea "Config" para más información.

`--non-interactive`

En el caso de un fallo de autenticación, o credenciales insuficientes, evita que se soliciten las credenciales (ej: nombre de usuario o clave). Esto es útil si está ejecutando Subversion dentro de un script automático y le resulta más apropiado que Subversion falle en lugar de pedir más información de manera interactiva.

`--non-recursive` (`-N`)

Evita que un subcomando entre de manera recursiva en subdirectorios. La mayoría de los subcomandos son recursivos por defecto, pero algunos subcomandos—normalmente aquellos que tienen el potencial de borrar o deshacer sus modificaciones locales—no lo son.

`--notice-ancestry`

Presta atención a la ascendencia cuando calcula diferencias.

`--old` *`ARG`*

Usa *`ARG`* como el objetivo antiguo.

`--password` *`PASS`*

Indica en la línea de comando la palabra clave para la autenticación—de otro modo, siempre que sea necesario, Subversion le preguntará por ella.

`--quiet` (`-q`)

Solicita que el cliente imprima solamente la información esencial mientras realiza su operación.

`--recursive` (`-R`)

Obliga a un subcomando a que entre de manera recursiva en subdirectorios. La mayoría de los subcomandos son recursivos por defecto.

`--relocate` *`FROM TO [PATH...]`*

Usado con el subcomando **svn switch**, cambia la ubicación del repositorio que su copia local de trabajo referencia. Esto es útil si la ubicación de su repositorio cambia y tiene una copia local existente que querría continuar usando. Consulte **svn switch** para ver un ejemplo.

`--revision` (`-r`) *`REV`*

Indica que va a proporcionar una revisión (o rango de revisiones) para una operación en particular. Puede proporcionar números de revisión, palabras clave o fechas (rodeadas por llaves), como argumentos del parámetro de revisión. Si quiere proporcionar un rango de revisiones, puede proporcionar dos revisiones separadas por dos puntos. Por ejemplo:

```
$ svn log -r 1729
$ svn log -r 1729:HEAD
$ svn log -r 1729:1744
$ svn log -r {2001-12-04}:{2002-02-17}
$ svn log -r 1729:{2002-02-17}
```

Vea “Palabras clave de la revisión” para obtener más información.

`--revprop`

Opera sobre una propiedad de revisión en lugar de una propiedad Subversion particular de un fichero o directorio. Este parámetro requiere que también indique la revisión con el parámetro `--revision` (`-r`). Vea “Propiedades no versionadas” para más detalles sobre propiedades no versionadas.

`--show-updates` (`-u`)

Hace que el cliente muestre la información sobre qué ficheros de su copia local de trabajo no están actualizados. Esto realmente no actualiza ninguno de sus ficheros—solamente muestra qué ficheros serían actualizados en caso de ejecutar **svn update**.

`--stop-on-copy`

Hace que el subcomando de Subversion que está recorriendo la historia de un recurso versionado deje de procesar esa información histórica cuando encuentre una copia—es decir, un punto en la historia cuando el recurso fue copiado desde otra ubicación del repositorio.

`--strict`

Hace que Subversion siga unas semánticas estrictas, una noción que es bastante vaga a no ser que hablemos de subcomandos específicos.

`--targets` *`FILENAME`*

Indica a Subversion que obtenga un listado de los ficheros sobre los que usted desea operar a partir del fichero proporcionado en lugar de listar todos sus nombres en la línea de comando.

`--username` *`NAME`*

Indica que está proporcionando su nombre de usuario para la autenticación en la línea de comando—de lo contrario, si es necesario, Subversion le preguntará por el.

`--verbose` (`-v`)

Solicita que el cliente imprima tanta información como sea posible mientras ejecuta cualquier subcomando. Esto puede tener como resultado que Subversion imprima campos adicionales, información detallada sobre cada fichero, o información adicional relacionada con las acciones que toma.

`--version`

Imprime la información de versión del cliente. Esta información no sólo incluye el número del cliente, sino que también muestra un listado de todos los módulos de acceso a repositorio que el cliente puede usar para acceder a repositorios de Subversion.

`--xml`

Imprime la salida en formato XML.

Subcomandos de svn

Nombre

svn add — Añade ficheros y directorios.

Sinopsis

```
svn add PATH...
```

Descripción

Añade ficheros y directorios a su copia de trabajo local y programa su adición al repositorio. Serán copiados al repositorio la próxima vez que envíe todos sus cambios al repositorio. Si añade algo o cambia de parecer antes del envío, puede desprogramar la adición usando **svn revert**.

Nombres alternativos

Ninguno

Cambios

En copia local

Accede al repositorio

No

Parámetros

```
--targets FILENAME
--non-recursive (-N)
--quiet (-q)
--config-dir DIR
--auto-props
--no-auto-props
```

Ejemplos

Para añadir un fichero a su copia de trabajo local:

```
$ svn add foo.c
A         foo.c
```

Cuando añade un directorio, el comportamiento por defecto de **svn add** es recurrir dentro:

```
$ svn add testdir
A         testdir
A         testdir/a
A         testdir/b
A         testdir/c
A         testdir/d
```

Puede añadir un directorio sin añadir su contenido:

```
$ svn add --non-recursive otherdir
A         otherdir
```

Nombre

svn blame — Muestra en línea información del autor y la revisión sobre los ficheros o URLs especificados.

Sinopsis

```
svn blame TARGET...
```

Descripción

Muestra en línea información del autor y la revisión sobre los ficheros o URLs especificados. Cada línea de texto es anotada al principio con el autor (nombre de usuario) y el número de revisión para el último cambio de esa línea.

Nombres alternativos

praise, annotate, ann

Cambios

Ninguno

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Si quiere ver el código fuente de readme.txt en su repositorio de pruebas anotado con la información de blame:

```
$ svn blame http://svn.red-bean.com/repos/test/readme.txt
     3      sally This is a README file.
     5      harry You should read this.
```

Nombre

svn cat — Vuelca el contenido de los ficheros o URLs especificados.

Sinopsis

```
svn cat TARGET...
```

Descripción

Vuelca el contenido de los ficheros o URLs especificados. Para obtener los contenidos de directorios, vea **svn list**.

Nombres alternativos

Ninguno

Cambios

Ninguno

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Si quiere ver readme.txt de su repositorio sin obtener una copia local:

```
$ svn cat http://svn.red-bean.com/repos/test/readme.txt
This is a README file.
You should read this.
```

Si su copia local no está actualizada (o tiene modificaciones locales) y quiere ver la revisión HEAD de un fichero de su copia local, **svn cat** obtendrá automáticamente la revisión HEAD cuando especifica una ruta:

```
$ cat foo.c
This file is in my local working copy
and has changes that I've made.

$ svn cat foo.c
Latest revision fresh from the repository!
```

Nombre

svn checkout — Obtiene una copia local de trabajo de un repositorio.

Sinopsis

```
svn checkout URL... [PATH]
```

Descripción

Obtiene una copia local de trabajo de un repositorio. Si omite *PATH*, el nombre base de la URL será usado como el destino. Si proporciona múltiples URLS, cada una será obtenida en un subdirectorio de PATH, con el nombre del subdirectorio como nombre base de la URL.

Nombres alternativos

co

Cambios

Crea una copia local de trabajo.

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--quiet (-q)
--non-recursive (-N)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Obtener una copia local en un directorio llamado 'mine':

```
$ svn checkout file:///tmp/repos/test mine
A  mine/a
A  mine/b
Checked out revision 2.
$ ls
mine
```

Obtener dos directorios diferentes en dos copias locales de trabajo separadas:

```
$ svn checkout file:///tmp/repos/test  file:///tmp/repos/quiz
A  test/a
A  test/b
```

```
Checked out revision 2.
A  quiz/l
A  quiz/m
Checked out revision 2.
$ ls
quiz  test
```

Obtener dos directorios diferentes en dos copias locales de trabajo separadas, pero guardando ambas en un directorio llamado 'working copies':

```
$ svn checkout file:///tmp/repos/test  file:///tmp/repos/quiz working-copies
A  working-copies/test/a
A  working-copies/test/b
Checked out revision 2.
A  working-copies/quiz/l
A  working-copies/quiz/m
Checked out revision 2.
$ ls
working-copies
```

Si interrumpe la operación (o alguna otra cosa interrumpe su operación de descarga como una pérdida de conectividad, etc), puede continuarla ya sea ejecutando el mismo comando para obtener la copia local de nuevo, o actualizando la copia local de trabajo incompleta:

```
$ svn checkout file:///tmp/repos/test test
A  test/a
A  test/b
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn checkout file:///tmp/repos/test test
A  test/c
A  test/d
^C
svn: The operation was interrupted
svn: caught SIGINT

$ cd test
$ svn update
A  test/e
A  test/f
Updated to revision 3.
```

Nombre

svn cleanup — Limpia la copia local de trabajo de forma recursiva.

Sinopsis

```
svn cleanup [PATH...]
```

Descripción

Limpia la copia local de trabajo de forma recursiva, eliminando bloqueos para continuar operaciones inacabadas. Si alguna vez obtiene el error “copia local bloqueada”, ejecute este comando para eliminar viejos bloqueos y llevar su copia local de trabajo de nuevo a un estado usable. Vea Apéndice B, *Solución de problemas*.

Si, por alguna razón, un comando **svn update** falla a causa de un problema relacionado con la ejecución de un programa de diferenciación externo (ej: error de entrada del usuario o de la red), pase el parámetro `--diff3-cmd` para que el proceso de limpiado complete fusionados pendientes con su programa de diferenciación externo. También puede especificar un directorio de configuración con el parámetro `--config-dir`, pero debería necesitar estos parámetros con extrema infrecuencia.

Nombres alternativos

Ninguno

Cambios

En copia local

Accede al repositorio

No

Parámetros

```
--diff3-cmd CMD
--config-dir DIR
```

Ejemplos

Bueno, no hay muchos ejemplos aquí dado que **svn cleanup** no genera salida alguna. Si no indica una ruta, se usará “`.`”.

```
$ svn cleanup

$ svn cleanup /path/to/working-copy
```

Nombre

svn commit — Envía cambios desde su copia de trabajo local al repositorio.

Sinopsis

```
svn commit [PATH...]
```

Descripción

Envía cambios desde su copia de trabajo local al repositorio. Si no proporciona un mensaje de cambios en la operación, ya sea usando los parámetros `--file` o `--message`, **svn** abrirá su editor para que componga el mensaje del informe de cambios. Vea la sección `editor-cmd` en “Config”.

Si comienza un envío y Subversion lanza su editor para componer el mensaje de cambios, todavía puede abortar sin enviar los cambios. Si quiere cancelar su envío, simplemente salga de su editor sin salvar el mensaje de cambios y Subversion le preguntará si quiere abortar el envío, continuar sin mensaje alguno, o editar de nuevo el mensaje.

Nombres alternativos

ci (abreviación de “check in” y no “co”, que es la abreviación de “checkout”)

Cambios

Copia local de trabajo, repositorio

Accede al repositorio

Si

Parámetros

```
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--non-recursive (-N)
--targets FILENAME
--force-log
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
--config-dir DIR
```

Ejemplos

Enviar al servidor una modificación simple a un fichero con el mensaje del informe de cambios en la línea de comando y el objetivo implícito del directorio actual (“`.`”):

```
$ svn commit -m "added howto section."
Sending        a
Transmitting file data .
```

```
Committed revision 3.
```

Enviar los cambios de una modificación al fichero `foo.c` (especificada de forma explícita en la línea de comando con el mensaje del informe de cambios contenido en un fichero llamado `msg`:

```
$ svn commit -F msg foo.c
Sending        foo.c
Transmitting file data .
Committed revision 5.
```

Si quiere usar un fichero que está bajo control de versiones para el mensaje del informe de cambios con `--file`, necesita pasar también el parámetro `--force-log`:

```
$ svn commit --file file_under_vc.txt foo.c
svn: The log message file is under version control
svn: Log message file is a versioned file; use '--force-log' to override

$ svn commit --force-log --file file_under_vc.txt foo.c
Sending        foo.c
Transmitting file data .
Committed revision 6.
```

Para enviar al repositorio el cambio de un fichero programado para ser borrado:

```
$ svn commit -m "removed file 'c'."
Deleting       c

Committed revision 7.
```

Nombre

svn copy — Copia un fichero o un directorio en la copia de trabajo local o en el repositorio.

Sinopsis

```
svn copy SRC DST
```

Descripción

Copia un fichero en la copia de trabajo local o en el repositorio. *`SRC`* y *`DST`* pueden ser ambos o bien la ruta de una copia de trabajo local (WC) o una URL:

WC -> WC

Copia y programa la adición (con historia) de un elemento.

WC -> URL

Envía al servidor inmediatamente una copia de WC a URL.

URL -> WC

Obtiene una copia local de URL en WC, y la programa para una adición.

URL -> URL

Copia realizada por completo en el servidor. Esto normalmente se usa para crear una rama o etiqueta.

Sólo puede copiar ficheros dentro de un mismo repositorio. Subversion no permite realizar copias entre repositorios.

Nombres alternativos

cp

Cambios

En repositorio si el destino es una URL.

En copia local si el destino es una ruta WC.

Accede al repositorio

Si la fuente o el destino están en el repositorio, o si tiene que consultar el número de revisión de la fuente.

Parámetros

```
--message (-m) TEXT
--file (-F) FILE
--revision (-r) REV
--quiet (-q)
--username USER
--password PASS
```

```
--no-auth-cache
--non-interactive
--force-log
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
```

Ejemplos

Copiar un elemento dentro de su copia de trabajo local (únicamente programa la copia—nada ocurre en el repositorio hasta que envíe los cambios):

```
$ svn copy foo.txt bar.txt
A         bar.txt
$ svn status
A  +   bar.txt
```

Copiar un elemento desde su copia de trabajo local a una URL en el repositorio (envía el cambio inmediatamente, así que debe proporcionar el mensaje para el informe de cambios):

```
$ svn copy near.txt file:///tmp/repos/test/far-away.txt -m "Remote copy."

Committed revision 8.
```

Copiar un elemento desde el repositorio a su copia de trabajo local (únicamente programa la copia—nada ocurre en el repositorio hasta que envíe los cambios):

¡Este es el método recomendado para recuperar un fichero borrado en su repositorio!

```
$ svn copy file:///tmp/repos/test/far-away near-here
A         near-here
```

Y finalmente, una copia entre dos URLs:

```
$ svn copy file:///tmp/repos/test/far-away file:///tmp/repos/test/over-there -m "remote
copy."

Committed revision 9.
```

Este es el modo más sencillo para “etiquetar” una revisión en su repositorio—simplemente haga **svn copy** de esa revisión (habitualmente HEAD) en su directorio de etiquetas.

```
$ svn copy file:///tmp/repos/test/trunk file:///tmp/repos/test/tags/0.6.32-prerelease
-m "tag tree"

Committed revision 12.
```

Y no se preocupe si se olvidó de etiquetar—siempre puede especificar una revisión anterior y etiquetarla en cualquier momento:

```
$ svn copy -r 11 file:///tmp/repos/test/trunk
file:///tmp/repos/test/tags/0.6.32-prerelease -m "Forgot to tag at rev 11"

Committed revision 13.
```

Nombre

svn delete — Borra un elemento de una copia de trabajo local o del repositorio.

Sinopsis

```
svn delete PATH...

svn delete URL...
```

Descripción

Los elementos especificados por *PATH* son programados para ser borrados en el siguiente envío al repositorio. Los ficheros (y directorios que no han sido enviados al repositorio) son borrados inmediatamente de la copia de trabajo local. El comando no eliminará elementos no versionados o modificados; use el parámetro `--force` para invalidar este comportamiento.

Los elementos especificados por una URL son eliminados del repositorio inmediatamente. Múltiples URLs se eliminan en una única transacción atómica.

Nombres alternativos

del, remove, rm

Cambios

Copia local de trabajo si se opera sobre ficheros, repositorio si se opera sobre URLs

Accede al repositorio

Sólo si se opera sobre URLs

Parámetros

```
--force
--force-log
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--targets FILENAME
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
```

Ejemplos

Usar **svn** para borrar un fichero de su copia local de trabajo meramente lo programa para ser borrado. Cuando envía cambios al repositorio, el fichero es borrado del mismo.

```
$ svn delete myfile
D         myfile
```

```
$ svn commit -m "Deleted file 'myfile'."
Deleting       myfile
Transmitting file data .
Committed revision 14.
```

Borrar una URL es, no obstante, inmediato, por lo que debe proporcionar un mensaje para el informe de cambios:

```
$ svn delete -m "Deleting file 'yourfile'" file:///tmp/repos/test/yourfile

Committed revision 15.
```

Aquí tiene un ejemplo de cómo forzar el borrado de un fichero que tiene modificaciones locales:

```
$ svn delete over-there
svn: Attempting restricted operation for modified resource
svn: Use --force to override this restriction
svn: 'over-there' has local modifications

$ svn delete --force over-there
D         over-there
```

Nombre

svn diff — Muestra las diferencias entre dos rutas.

Sinopsis

```
svn diff [-r N[:M]] [--old OLD-TGT] [--new NEW-TGT] [PATH...]

svn diff -r N:M URL

svn diff [-r N[:M]] URL1[@N] URL2[@M]
```

Descripción

Muestra las diferencias entre dos rutas. Puede usar **svn diff** de tres modos diferentes:

svn diff [-r N[:M]] [--old OLD-TGT] [--new NEW-TGT] [PATH...] muestra las diferencias entre *OLD-TGT* y *NEW-TGT*. Si proporciona *PATH*s, éstas rutas son tratadas como relativas a *OLD-TGT* y *NEW-TGT* y la salida se restringe a diferencias en sólo aquellas rutas. *OLD-TGT* y *NEW-TGT* pueden ser rutas de copias locales de trabajo o *URL*[@*REV*]. *OLD-TGT* apunta por defecto al directorio de trabajo actual y *NEW-TGT* apunta por defecto a *OLD-TGT*. *N* es por defecto BASE o, si *OLD-TGT* es una URL, HEAD. *M* es por defecto la versión actual o, si *NEW-TGT* es una URL, HEAD. **svn diff -r N** ajusta la revisión de *OLD-TGT* a *N*, **svn diff -r N:M** también ajusta la revisión de *NEW-TGT* a *M*.

svn diff -r N:M URL es la versión abreviada de **svn diff -r N:M --old=URL --new=URL**.

svn diff [-r N[:M]] URL1[@N] URL2[@M] es la versión abreviada de **svn diff [-r N[:M]] --old=URL1 --new=URL2**.

Si *TARGET* es una URL, entonces las revisiones N y M pueden ser proporcionadas vía `--revision` o usando la notación "@" descrita anteriormente.

Si *TARGET* es una copia de trabajo local, entonces el parámetro `--revision` significa:

--revision N:M

El servidor compara *TARGET@N* y *TARGET@M*.

--revision N

El cliente compara *TARGET@N* contra la copia de trabajo local.

(no --revision)

El cliente compara la versión base con la versión de la copia local de trabajo de *TARGET*.

Si se usa la sintaxis alternativa, el servidor compara *URL1* y *URL2* en las revisiones *N* y *M* respectivamente. Si omite *N* o *M*, se asume el valor de HEAD.

Por defecto, **svn diff** ignora la ascendencia de los ficheros y meramente compara el contenido entre dos ficheros. Si usa `--notice-ancestry`, la ascendencia de las rutas especificadas será tomada en consideración durante la comparación de revisiones (es decir, si ejecuta **svn diff** sobre dos ficheros de contenido idéntico pero diferente ascendencia verá el contenido entero del fichero como si hubiese sido borrado y añadido de nuevo).

Nombres alternativos

di

Cambios

Ninguno

Accede al repositorio

Para obtener las diferencias contra cualquier cosa menos la revisión BASE de su copia local de trabajo

Parámetros

```
--revision (-r) REV
--old OLD-TARGET
--new NEW-TARGET
--extensions (-x) "ARGS"
--non-recursive (-N)
--diff-cmd CMD
--notice-ancestry
--username USER
--password PASS
--no-auth-cache
--non-interactive
--no-diff-deleted
--config-dir DIR
```

Ejemplos

Para comparar la versión BASE con su copia local de trabajo (uno de los usos más populares de **svn diff**):

```
$ svn diff COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 4404)
+++ COMMITTERS (working copy)
```

Compruebe cómo sus modificaciones locales son comparadas contra una versión anterior:

```
$ svn diff -r 3900 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3900)
+++ COMMITTERS (working copy)
```

Comparar la revisión 3000 con la 3500 usando la sintaxis “@”:

```
$ svn diff http://svn.collab.net/repos/svn/trunk/COMMITTERS@3000
http://svn.collab.net/repos/svn/trunk/COMMITTERS@3500
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
…
```

Comparar la revisión 3000 con la 3500 usando la notación de rango (en este caso sólo proporciona una URL):

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk/COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
```

Comparar la revisión 3000 con la 3500 de todos los ficheros en `trunk` usando la notación de rango:

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk
```

Comparar la revisión 3000 con la 3500 de sólo tres ficheros en `trunk` usando la notación de rango:

```
$ svn diff -r 3000:3500 --old http://svn.collab.net/repos/svn/trunk COMMITTERS README
HACKING
```

Si tiene una copia de trabajo local, puede obtener las diferencias sin necesidad de teclear largas URLs:

```
$ svn diff -r 3000:3500 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (revision 3000)
+++ COMMITTERS (revision 3500)
```

Use la opción `--diff-cmd` *`CMD`* `-x` para pasar argumentos de manera directa al programa externo de diferenciación

```
$ svn diff --diff-cmd /usr/bin/diff -x "-i -b" COMMITTERS
Index: COMMITTERS
===================================================================
0a1,2
> This is a test
>
```

Nombre

svn export — Exporta un árbol de directorios limpio.

Sinopsis

```
svn export [-r REV] URL [PATH]

svn export PATH1 PATH2
```

Descripción

La primera forma exporta un árbol de directorios limpio desde el repositorio especificado por URL, en la revisión *REV* si ésta es proporcionada, en caso contrario se usará HEAD, en *PATH*. Si omite *PATH*, el último componente de *URL* será usado para el nombre del directorio local.

La segunda forma exporta un árbol de directorios limpio desde una copia de trabajo local especificada por *PATH1* en *PATH2*. Todos los cambios locales serán preservados, pero los ficheros que no estén bajo control de versiones no serán copiados.

Nombres alternativos

Ninguno

Cambios

Discos locales

Accede al repositorio

Sólo si se exporta desde una URL

Parámetros

```
--revision (-r) REV
--quiet (-q)
--force
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Exportar desde su copia de trabajo local (no muestra cada fichero y directorio individual):

```
$ svn export a-wc my-export
Export complete.
```

Exportar desde un repositorio (muestra cada fichero y directorio):

```
$ svn export file:///tmp/repos my-export
```

```
A  my-export/test
A  my-export/quiz
…
Exported revision 15.
```

Nombre

svn help — ¡Ayuda!

Sinopsis

```
svn help [SUBCOMMAND...]
```

Descripción

¡Este es su mejor amigo cuando usa Subversion y este libro no está a su alcance!

Nombres alternativos

?, h

Cambios

Ninguno

Accede al repositorio

No

Parámetros

```
--version
--quiet (-q)
```

Nombre

svn import — Envía de forma recursiva los cambios de PATH en URL.

Sinopsis

```
svn import [PATH] URL
```

Descripción

Envía de forma recursiva los cambios de *PATH* a *URL*. Si se omite *PATH* se asumirá "`.`". Los directorios padre son creados en el repositorio según son necesarios.

Nombres alternativos

Ninguno

Cambios

En repositorio

Accede al repositorio

Si

Parámetros

```
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--non-recursive (-N)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--force-log
--editor-cmd EDITOR
--encoding ENC
--config-dir DIR
--auto-props
--no-auto-props
```

Ejemplos

Esto copia el directorio local `myproj` en la raíz de su repositorio:

```
$ svn import -m "New import" myproj http://svn.red-bean.com/repos/test
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 16.
```

Esto copia el directorio local `myproj` en el directorio `trunk/vendors` de su repositorio. No es necesario que `trunk/vendors` exista antes de realizar la copia—**svn import** creará de forma recursiva los directorios necesarios por usted:

```
$ svn import -m "New import" myproj \
    http://svn.red-bean.com/repos/test/trunk/vendors/myproj
Adding         myproj/sample.txt
…
Transmitting file data .........
Committed revision 19.
```

Tras copiar los datos, fíjese que el árbol original *no* está bajo control de versiones. Antes de comenzar a trabajar, necesita obtener una copia de trabajo local fresca del árbol mediante **svn checkout**.

Nombre

svn info — Imprime información sobre RUTAs.

Sinopsis

```
svn info [PATH...]
```

Descripción

Imprime información sobre las rutas de su copia local de trabajo, incluyendo:

- Ruta
- Nombre
- URL
- Revisión
- Tipo de nodo
- Autor del último cambio
- Revisión del último cambio
- Fecha del último cambio
- Texto actualizado por última vez
- Propiedades actualizadas por última vez
- Suma de verificación

Nombres alternativos

Ninguno

Cambios

Ninguno

Accede al repositorio

No

Parámetros

```
--targets FILENAME
--recursive (-R)
--config-dir DIR
```

Ejemplos

svn info mostrará toda la información útil que tiene para los elementos en su copia local de trabajo. Mostrará información sobre ficheros:

```
$ svn info foo.c
Path: foo.c
Name: foo.c
URL: http://svn.red-bean.com/repos/test/foo.c
Revision: 4417
Node Kind: file
Schedule: normal
Last Changed Author: sally
Last Changed Rev: 20
Last Changed Date: 2003-01-13 16:43:13 -0600 (Mon, 13 Jan 2003)
Text Last Updated: 2003-01-16 21:18:16 -0600 (Thu, 16 Jan 2003)
Properties Last Updated: 2003-01-13 21:50:19 -0600 (Mon, 13 Jan 2003)
Checksum: /3L38YwzhT93BWvgpdF6Zw==
```

Y también mostrará información sobre directorios:

```
$ svn info vendors
Path: trunk
URL: http://svn.red-bean.com/repos/test/vendors
Revision: 19
Node Kind: directory
Schedule: normal
Last Changed Author: harry
Last Changed Rev: 19
Last Changed Date: 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003)
```

Nombre

svn list — Lista las entradas de directorio en el repositorio.

Sinopsis

```
svn list [TARGET...]
```

Descripción

Lista cada fichero *`OBJETIVO`* y los contenidos de cada directorio *`OBJETIVO`* tal y como existen en el repositorio. Si *`OBJETIVO`* es una ruta de una copia de trabajo local, se usará la URL de repositorio correspondiente.

El *`OBJETIVO`* por defecto es "`.`", lo que equivale a la URL del repositorio correspondiente al directorio actual en la copia local de trabajo.

Con `--verbose`, se muestran los siguientes campos del estado de los elementos:

- Número de revisión del último envío de cambios al repositorio
- Autor del último envío de cambios al repositorio
- Tamaño (en bytes)
- Fecha y marca de tiempo del último envío de cambios al repositorio

Nombres alternativos

ls

Cambios

Ninguno

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--verbose (-v)
--recursive (-R)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

svn list es útil sobre todo cuando desea ver qué ficheros contiene un repositorio sin tener que descargar una copia local de trabajo del mismo:

```
$ svn list http://svn.red-bean.com/repos/test/support
README.txt
INSTALL
examples/
…
```

Al igual que el comando UNIX **ls**, puede pasar el parámetro `--verbose` para obtener información adicional:

```
$ svn list --verbose file:///tmp/repos
     16 sally         28361 Jan 16 23:18 README.txt
     27 sally             0 Jan 18 15:27 INSTALL
     24 harry               Jan 18 11:27 examples/
```

Para más detalles, vea “**svn list**”.

Nombre

svn log — Muestra los mensajes de los informes de cambios.

Sinopsis

```
svn log [PATH]

svn log URL [PATH...]
```

Descripción

El objetivo por defecto es la ruta de su directorio actual. Si no proporciona parámetro alguno, **svn log** muestra los mensajes de los informes de cambios de todos los ficheros y directorios dentro del (e incluyendo) directorio actual de trabajo en su copia local. Puede refinar los resultados especificando una ruta, una o más revisiones, o cualquier combinación de ambas. El rango de revisión por defecto para una ruta local es `BASE:1`.

Si especifica una URL sola, entonces imprime los mensajes de los informes de cambios para todo lo que contenga esa URL. Si añade rutas tras la URL, sólo los mensajes de esas rutas bajo esa URL serán mostrados. El rango de revisión por defecto de una URL es `HEAD:1`.

Con `--verbose`, **svn log** también mostrará todas las rutas afectadas por cada mensaje de informe de cambios. Con `--quiet`, **svn log** no mostrará el cuerpo del mensaje del informe de cambios (esto es compatible con `--verbose`).

Cada mensaje de informe de cambios es impreso una única vez, incluso si solicitó de forma explícita más de una ruta afectadas por la misma revisión. Por defecto se sigue la historia de copiado de los ficheros. Use `--stop-on-copy` para desactivar este comportamiento, lo cual puede ser útil para encontrar puntos de creación de ramas.

Nombres alternativos

Ninguno

Cambios

Ninguno

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--quiet (-q)
--verbose (-v)
--targets FILENAME
--stop-on-copy
--incremental
--xml
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Puede ver los mensajes de informes de cambios para todas las rutas que fueron modificadas en su copia local ejecutando **svn log**:

```
$ svn log
------------------------------------------------------------------------
r20 | harry | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

Tweak.
------------------------------------------------------------------------
r17 | sally | 2003-01-16 23:21:19 -0600 (Thu, 16 Jan 2003) | 2 lines
…
```

Examinar todos los mensajes de informes de cambios para un fichero particular de su copia local de trabajo:

```
$ svn log foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Si no tiene a mano una copia local, puede obtener los mensajes de informes de cambios a partir de una URL:

```
$ svn log http://svn.red-bean.com/repos/test/foo.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Si quiere usar varias rutas distintas bajo una misma URL, puede usar la sintaxis `URL [PATH...]`.

```
$ svn log http://svn.red-bean.com/repos/test/ foo.c bar.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (Mon, 13 Jan 2003) | 1 line

Added defines.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (Fri, 10 Jan 2003) | 1 line

Added new file bar.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (Tue, 07 Jan 2003) | 3 lines
…
```

Eso es igual que poner de forma explícita ambas URLs en la línea de comando:

```
$ svn log http://svn.red-bean.com/repos/test/foo.c \
          http://svn.red-bean.com/repos/test/bar.c
…
```

Cuando concatene los resultados de múltiples llamadas al comando log, quizás desee usar el parámetro `--incremental`. **svn log** normalmente imprime una línea de guiones al comienzo de cada mensaje de informe de cambios, entre mensajes, y después el último. Si ejecuta **svn log** sobre un rango de dos revisiones, obtendría esto:

```
$ svn log -r 14:15
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
r15 | ...

------------------------------------------------------------------------
```

No obstante, si quisiese recoger dos mensajes de informes de cambios no secuenciales en un mismo fichero, podría hacer algo como esto:

```
$ svn log -r 14 > mylog
$ svn log -r 19 >> mylog
$ svn log -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
------------------------------------------------------------------------
r19 | ...

------------------------------------------------------------------------
------------------------------------------------------------------------
r27 | ...

------------------------------------------------------------------------
```

Puede evitar la duplicidad de líneas de guiones en la salida usando el parámetro incremental:

```
$ svn log --incremental -r 14 > mylog
$ svn log --incremental -r 19 >> mylog
$ svn log --incremental -r 27 >> mylog
$ cat mylog
------------------------------------------------------------------------
r14 | ...

------------------------------------------------------------------------
r19 | ...

------------------------------------------------------------------------
r27 | ...
```

El parámetro `--incremental` proporciona un control de la salida similar cuando usa el parámetro `--xml`.

Si ejecuta **svn log** sobre una ruta específica e indica una revisión específica pero no obtiene ninguna salida en absoluto

```
$ svn log -r 20 http://svn.red-bean.com/untouched.txt
------------------------------------------------------------------------
```

Eso sólo significa que esa ruta no fue modificada en esa revisión. Si ejecuta estos comandos desde la raíz del repositorio, o sabe qué fichero cambió en aquella revisión, puede especificarlo de forma explícita:

```
$ svn log -r 20 touched.txt
------------------------------------------------------------------------
r20 | sally | 2003-01-17 22:56:19 -0600 (Fri, 17 Jan 2003) | 1 line

Made a change.
------------------------------------------------------------------------
```

Nombre

svn merge — Aplica las diferencias entre dos fuentes a una ruta de su copia local de trabajo.

Sinopsis

```
svn merge sourceURL1[@N] sourceURL2[@M] [WCPATH]

svn merge -r N:M SOURCE [PATH]
```

Descripción

En su primera forma, las URLs fuente son acotadas a las revisiones *N* y *M*. Éstas son las dos fuentes que serán comparadas. Por defecto las revisiones son HEAD si se omiten.

En su segunda forma, *SOURCE* puede ser una URL o un elemento de su copia local de trabajo, en cuyo caso se usará su URL correspondiente. Esta URL, en sus revisiones *N* y *M*, define las dos fuentes que serán comparadas.

WCPATH es la ruta de la copia local de trabajo que recibirá los cambios. Si se omite *WCPATH*, se asumirá un valor por defecto de "`.`", a no ser que que las fuentes tengan nombres base idénticos que concuerden con un fichero dentro de "`.`": en cuyo caso, las diferencias serán aplicadas a aquel fichero.

A diferencia de **svn diff**, el comando de fusionado toma en consideración la ascendencia de los ficheros cuando realiza sus operaciones. Esto es muy importante cuando está fusionando cambios provenientes de una rama en otra y ha renombrado un fichero en una, pero no en la otra.

Nombres alternativos

Ninguno

Cambios

En copia local

Accede al repositorio

Sólo si trabaja con URLs

Parámetros

```
--revision (-r) REV
--non-recursive (-N)
--quiet (-q)
--force
--dry-run
--diff3-cmd CMD
--ignore-ancestry
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Fusionar una rama de nuevo en el tronco (asumiendo que tiene una copia local del tronco, y que la rama fue creada en la revisión 250):

```
$ svn merge -r 250:HEAD http://svn.red-bean.com/repos/branches/my-branch
U  myproj/tiny.txt
U  myproj/thhgttg.txt
U  myproj/win.txt
U  myproj/flo.txt
```

Si creó una rama en la revisión 23, y quiere fusionar los cambios del tronco en ella, puede hacer lo siguiente desde su copia local de trabajo de la rama:

```
$ svn merge -r 23:30 file:///tmp/repos/trunk/vendors
U  myproj/thhgttg.txt
…
```

Para fusionar cambios en un único fichero:

```
$ cd myproj
$ svn merge -r 30:31 thhgttg.txt
U  thhgttg.txt
```

Nombre

svn mkdir — Crea un nuevo directorio bajo control de versiones.

Sinopsis

```
svn mkdir PATH...

svn mkdir URL...
```

Descripción

Crea un directorio con el nombre proporcionado por el componente final de *PATH* o URL. Un directorio especificado por la ruta de una copia local de trabajo es programado para ser añadido en la copia local. Un directorio especificado por una URL es creado en el repositorio de manera inmediata. Múltiples directorios con URLs son creados de forma atómica. En ambos casos, todos los directorios intermedios deben existir previamente.

Nombres alternativos

Ninguno

Cambios

En la copia local de trabajo, en el repositorio si se opera sobre una URL

Accede al repositorio

Sólo cuando opera sobre una URL

Parámetros

```
--message (-m) TEXT
--file (-F) FILE
--quiet (-q)
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--force-log
--config-dir DIR
```

Ejemplos

Crear un directorio en su copia local de trabajo:

```
$ svn mkdir newdir
A         newdir
```

Creación de un directorio en el repositorio (la creación es inmediata, por lo que se requiere un mensaje de informe de cambios):

```
$ svn mkdir -m "Making a new dir." http://svn.red-bean.com/repos/newdir
Committed revision 26.
```

Nombre

svn move — Mover un fichero o directorio.

Sinopsis

```
svn move SRC DST
```

Descripción

Este comando mueve un fichero o directorio en su copia local de trabajo o en el repositorio.

Este comando es equivalente a un **svn copy** seguido de un **svn delete**.

Subversion no soporta operaciones de movimiento entre copias locales y URLs. Adicionalmente, sólo puede mover ficheros dentro de un único repositorio—Subversion no soporta movimientos entre repositorios.

WC -> WC

Mueve y programa un fichero o directorio para que sea añadido (con historia).

URL -> URL

Renombrado que se realiza por completo en el lado del servidor.

Nombres alternativos

mv, rename, ren

Cambios

Copia local de trabajo, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--message (-m) TEXT
--file (-F) FILE
--revision (-r) REV
--quiet (-q)
--force
--username USER
--password PASS
--no-auth-cache
--non-interactive
--editor-cmd EDITOR
--encoding ENC
--force-log
```

```
--config-dir DIR
```

Ejemplos

Mover un fichero en su copia local de trabajo:

```
$ svn move foo.c bar.c
A         bar.c
D         foo.c
```

Mover un fichero en el repositorio (el cambio es inmediato, por lo que se requiere un mensaje de informe de cambios):

```
$ svn move -m "Move a file" http://svn.red-bean.com/repos/foo.c \
                            http://svn.red-bean.com/repos/bar.c

Committed revision 27.
```

Nombre

svn propdel — Borrar la propiedad de un elemento.

Sinopsis

```
svn propdel PROPNAME [PATH...]

svn propdel PROPNAME --revprop -r REV [URL]
```

Descripción

Esto elimina las propiedades de ficheros, directorios, o revisiones. La primera forma elimina propiedades versionadas en su copia local de trabajo, mientras que la segunda elimina propiedades remotas no versionadas en una revisión de repositorio.

Nombres alternativos

pdel, pd

Cambios

Copia local, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Borrar la propiedad de un fichero de su copia local de trabajo

```
$ svn propdel svn:mime-type  some-script
property 'svn:mime-type' deleted from 'some-script'.
```

Borrar la propiedad de una revisión:

```
$ svn propdel --revprop -r 26 release-date
property 'release-date' deleted from repository revision '26'
```

Nombre

svn propedit — Editar la propiedad de uno o más elementos bajo control de versiones.

Sinopsis

```
svn propedit PROPNAME PATH...

svn propedit PROPNAME --revprop -r REV [URL]
```

Descripción

Edite una o más propiedades usando su editor favorito. La primera forma permite editar propiedades versionadas en su copia local de trabajo, mientras que la segunda edita propiedades remotas no versionadas en una revisión de repositorio.

Nombres alternativos

pedit, pe

Cambios

Copia local, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--revision (-r) REV
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
--editor-cmd EDITOR
--config-dir DIR
```

Ejemplos

svn propedit facilita la modificación de propiedades que tienen múltiples valores:

```
$ svn propedit svn:keywords  foo.c
    <svn will launch your favorite editor here, with a buffer open
    containing the current contents of the svn:keywords property.  You
    can add multiple values to a property easily here by entering one
    value per line.>
Set new value for property 'svn:keywords' on 'foo.c'
```

Nombre

svn propget — Imprime el valor de una propiedad.

Sinopsis

```
svn propget PROPNAME [PATH...]

svn propget PROPNAME --revprop -r REV [URL]
```

Descripción

Imprime el valor de propiedades de ficheros, directorios o revisiones. La primera forma imprime la propiedad versionada de un elemento o elementos en su copia local de trabajo, mientras que la segunda imprime propiedades remotas no versionadas en una revisión de repositorio. Vea “Propiedades” para más información sobre propiedades.

Nombres alternativos

pget, pg

Cambios

Copia local, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--recursive (-R)
--revision (-r) REV
--revprop
--strict
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Examinar la propiedad de un fichero en su copia local de trabajo:

```
$ svn propget svn:keywords foo.c
Author
Date
Rev
```

Lo mismo sirve para una propiedad de revisión:

```
$ svn propget svn:log --revprop -r 20
```

```
Began journal.
```

Nombre

svn proplist — Mostrar todas las propiedades.

Sinopsis

```
svn proplist [PATH...]

svn proplist --revprop -r REV [URL]
```

Descripción

Muestra todas las propiedades de ficheros, directorios o revisiones. La primera forma muestra las propiedades versionadas de su copia local de trabajo, mientras que la segunda muestra las propiedades remotas no versionadas en una revisión de repositorio.

Nombres alternativos

plist, pl

Cambios

Copia local, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--verbose (-v)
--recursive (-R)
--revision (-r) REV
--quiet (-q)
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Puede usar proplist para ver las propiedades asociadas a un elemento en su copia local de trabajo:

```
$ svn proplist foo.c
Properties on 'foo.c':
  svn:mime-type
  svn:keywords
  owner
```

Pero con el parámetro `--verbose`, svn proplist es extremadamente útil ya que también muestra los valores de las propiedades:

```
$ svn proplist --verbose foo.c
Properties on 'foo.c':
  svn:mime-type : text/plain
  svn:keywords : Author Date Rev
  owner : sally
```

Nombre

svn propset — Asocia PROPNAME a PROPVAL en ficheros, directorios, o revisiones.

Sinopsis

```
svn propset PROPNAME [PROPVAL | -F VALFILE] PATH...

svn propset PROPNAME --revprop -r REV [PROPVAL | -F VALFILE] [URL]
```

Descripción

Asocia *PROPNAME* a *PROPVAL* en ficheros, directorios o revisiones. La primera forma crea un cambio versionado de una propiedad en su copia local de trabajo, la segunda forma crea un cambio no versionado de una propiedad en una revisión de repositorio.

Subversion tiene un número de propiedades “especiales” que afectan su comportamiento. Vea “Propiedades especiales” para más información sobre estas propiedades.

Nombres alternativos

pset, ps

Cambios

Copia local, repositorio si opera sobre una URL

Accede al repositorio

Sólo si opera sobre una URL

Parámetros

```
--file (-F) FILE
--quiet (-q)
--revision (-r) REV
--targets FILENAME
--recursive (-R)
--revprop
--username USER
--password PASS
--no-auth-cache
--non-interactive
--encoding ENC
--force
--config-dir DIR
```

Ejemplos

Establecer el tipo mime de un fichero:

```
$ svn propset svn:mime-type image/jpeg foo.jpg
property 'svn:mime-type' set on 'foo.jpg'
```

En sistemas UNIX, si quiere que un fichero tenga activado el permiso de ejecución:

```
$ svn propset svn:executable ON somescript
property 'svn:executable' set on 'somescript'
```

Quizás tenga una política interna de creación de propiedades para beneficio de sus colaboradores:

```
$ svn propset owner sally foo.c
property 'owner' set on 'foo.c'
```

Si cometió un error en un mensaje de informe de cambios de una revisión particular y desea corregirlo, use `--revprop` y cambie svn:log para que refleje el nuevo mensaje:

```
$ svn propset --revprop -r 25 svn:log "Journaled about trip to New York."
property 'svn:log' set on repository revision '25'
```

O, si no tiene una copia de trabajo local, puede proporcionar una URL.

```
$ svn propset --revprop -r 26 svn:log "Document nap." http://svn.red-bean.com/repos
property 'svn:log' set on repository revision '25'
```

Por último, puede decirle a propset que obtenga los datos de entrada de un fichero. Puede usar esto incluso para modificar el contenido de una propiedad a algo binario:

```
$ svn propset owner-pic -F sally.jpg moo.c
property 'owner-pic' set on 'moo.c'
```

Por defecto, no puede modificar propiedades de revisiones en un repositorio de Subversion. El administrador del repositorio debe activar de forma explícita la modificación de propiedades de revisión creando un gancho llamado `pre-revprop-change`. Vea “Scripts de enganche” para más información sobre scripts de enganche.

Nombre

svn resolved — Resuelve el estado “conflictivo” en ficheros o directorios de la copia de trabajo local.

Sinopsis

```
svn resolved PATH...
```

Descripción

Resuelve el estado “conflictivo” en ficheros o directorios de la copia de trabajo local. Esta rutina no resuelve marcas de conflicto a nivel semántico; meramente elimina los ficheros artefacto relacionados con el conflicto y permite que PATH pueda ser enviado de nuevo al repositorio; es decir, le dice a Subversion que los conflictos han sido “resueltos”. Vea “Resolver conflictos (fusionando los cambios de otros)” para una visión pormenorizada sobre la resolución de conflictos.

Nombres alternativos

Ninguno

Cambios

En copia local

Accede al repositorio

No

Parámetros

```
--targets FILENAME
--recursive (-R)
--quiet (-q)
--config-dir DIR
```

Ejemplos

Si obtiene un conflicto tras una actualización, su copia de trabajo local generará tres nuevos ficheros:

```
$ svn update
C  foo.c
Updated to revision 31.
$ ls
foo.c
foo.c.mine
foo.c.r30
foo.c.r31
```

Una vez haya resuelto el conflicto y `foo.c` esté listo para ser enviado al repositorio, ejecute **svn resolved** para indicarle a su copia de trabajo local que se ha encargado de todo.

Usted *puede* simplemente eliminar los ficheros del conflicto y enviar los cambios, pero **svn resolved** corrige algunos datos de gestión del área administrativa de su copia local aparte de eliminar los ficheros del conflicto, así que le recomendamos que use este comando.

Nombre

svn revert — Deshacer todas las modificaciones locales.

Sinopsis

```
svn revert PATH...
```

Descripción

Revierte cualquier cambio local sobre un fichero o directorio y resuelve cualquier estado de conflicto. **svn revert** no solamente revertirá el contenido de un elemento en su copia local de trabajo, sino también cualquier cambio de propiedades. Finalmente, puede usarlo para deshacer cualquier operación anteriormente programada (ej: ficheros programados para ser añadidos o borrados pueden ser “desprogramados”).

Nombres alternativos

Ninguno

Cambios

En copia local

Accede al repositorio

No

Parámetros

```
--targets FILENAME
--recursive (-R)
--quiet (-q)
--config-dir DIR
```

Ejemplos

Descartar cambios de un fichero:

```
$ svn revert foo.c
Reverted foo.c
```

Si quiere revertir todo un directorio de ficheros, use el parámetro `--recursive`:

```
$ svn revert --recursive .
Reverted newdir/afile
Reverted foo.c
Reverted bar.txt
```

Por último, puede deshacer cualquier operación programada:

```
$ svn add mistake.txt whoops
```

```
A         mistake.txt
A         whoops
A         whoops/oopsie.c

$ svn revert mistake.txt whoops
Reverted mistake.txt
Reverted whoops

$ svn status
?      mistake.txt
?      whoops
```

Si no proporciona objetivos a **svn revert**, no hará nada—para protegerle de perder accidentalmente los cambios de su copia local de trabajo, **svn revert** requiere que proporcione al menos un objetivo.

Nombre

svn status — Imprime el estado de los ficheros y directorios de su copia local de trabajo.

Sinopsis

```
svn status [PATH...]
```

Descripción

Imprime el estado de los ficheros y directorios de su copia local de trabajo. Sin argumentos, sólo imprime los elementos que han sido modificados localmente (sin acceder al repositorio). Con `--show-updates`, añade información sobre la revisión de trabajo y des-sincronización con el servidor. Con `--verbose`, imprime toda la información de revisión de cada elemento.

Las primeras cinco columnas de la salida miden cada una un carácter, y cada columna proporciona información sobre diferentes aspectos de cada elemento de su copia local de trabajo.

La primera columna indica si un elemento fue añadido, borrado, o modificado de algún modo.

' '

Sin modificaciones.

'A'

El elemento está programado para ser añadido.

'D'

El elemento está programado para ser borrado.

'M'

El elemento ha sido modificado.

'C'

El elemento está en conflicto tras recibir actualizaciones del repositorio.

'X'

El elemento está relacionado con una definición externa.

'I'

El elemento está siendo ignorado (ej: con la propiedad svn:ignore).

'?'

El elemento no está bajo control de versiones.

'!'

Falta el elemento (ej: lo ha movido o borrado sin usar **svn**). Esto también indica que un directorio es incompleto (la obtención del servidor o actualización del mismo fue interrumpida).

'~'

El elemento está versionado como un directorio, pero ha sido reemplazado por un fichero, o vice versa.

La segunda columna indica el estado de las propiedades del fichero o directorio.

' '

Sin modificaciones.

'M'

Las propiedades de este elemento han sido modificadas.

'C'

Las propiedades de este elemento están en conflicto con la actualización de propiedades recibida del repositorio.

La tercera columna sólo se rellena si la copia local de trabajo está bloqueada.

' '

Elemento no bloqueado.

'L'

Elemento bloqueado.

La cuarta columna sólo se rellena si el elemento está programado para ser añadido con historia.

' '

No se programó historia para el envío.

'+'

Historia programada para el envío.

La quinta columna sólo se rellena si la ubicación del elemento no está relacionada con su padre (vea “Cambiando la copia local de trabajo”).

' '

El elemento es hijo de su directorio padre.

'S'

El elemento es de otra ubicación.

La información de desfase aparece en la octava columna (sólo si pasa el parámetro `--show-updates`).

' '

El elemento en su copia loca de trabajo está actualizado.

'*'

Existe una nueva versión del elemento en el servidor.

Los campos restantes tienen una anchura variable y están delimitados por espacios. La revisión de trabajo es el siguiente campo si se pasan los parámetros `--show-updates` o `--verbose`.

Si pasa el parámetro `--verbose`, se muestra a continuación la última revisión enviada al repositorio junto con su autor.

La ruta de la copia local de trabajo es siempre el último campo, por lo que puede incluir espacios.

Nombres alternativos

stat, st

Cambios

Ninguno

Accede al repositorio

Sólo si usa `--show-updates`

Parámetros

```
--show-updates (-u)
--verbose (-v)
--non-recursive (-N)
--quiet (-q)
--no-ignore
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir
```

Ejemplos

Este es el modo más fácil para descubrir qué cambios ha realizado en su copia local de trabajo:

```
$ svn status wc
 M     wc/bar.c
A  +   wc/qax.c
```

Si quiere averiguar qué ficheros de su copia local de trabajo están desfasados, pase el parámetro `--show-updates` (esto *no* realizará cambio alguno en su copia local). Aquí puede ver que `wc/foo.c` ha cambiado en el repositorio desde la última vez que actualizamos nuestra copia local de trabajo:

```
$ svn status --show-updates wc
 M           965    wc/bar.c
       *     965    wc/foo.c
A  +         965    wc/qax.c
```

```
Status against revision:    981
```

`--show-updates` *solamente* muestra un asterisco junto a los elementos desfasados (es decir, elementos que serán actualizados desde el repositorio en cuanto ejecute **svn update**). `--show-updates` *no* hace que la lista de estado refleje la versión que hay del elemento en el repositorio.

Y finalmente, la mayor cantidad posible de información que puede obtener del subcomando de estado:

```
$ svn status --show-updates --verbose wc
 M           965       938 sally        wc/bar.c
       *     965       922 harry        wc/foo.c
A  +         965       687 harry        wc/qax.c
             965       687 harry        wc/zig.c
Head revision:   981
```

Para ver muchos más ejemplos de **svn status**, vea “**svn status**”.

Nombre

svn switch — Actualizar una copia local de trabajo a una URL diferente.

Sinopsis

```
svn switch URL [PATH]
```

Descripción

Este comando actualiza su copia local de trabajo a una nueva URL—habitualmente una URL que comparte un ancestro común con su copia de trabajo local, aunque no necesariamente. Este es el modo de Subversion de mover su copia de trabajo local a una nueva rama. Vea “Cambiando la copia local de trabajo” para un vistazo en profundidad sobre estas operaciones.

Nombres alternativos

sw

Cambios

En copia local

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
--non-recursive (-N)
--quiet (-q)
--diff3-cmd CMD
--relocate
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Si está actualmente dentro del directorio `vendors` el cual fue ramificado a `vendors-with-fix` y desea cambiar su copia local de trabajo a esta rama:

```
$ svn switch http://svn.red-bean.com/repos/branches/vendors-with-fix .
U  myproj/foo.txt
U  myproj/bar.txt
U  myproj/baz.c
U  myproj/qux.c
Updated to revision 31.
```

Y para cambiar de vuelta, simplemente proporcione la URL de la ubicación del repositorio del cual originalmente obtuvo su copia local de trabajo:

```
$ svn switch http://svn.red-bean.com/repos/trunk/vendors .
U  myproj/foo.txt
U  myproj/bar.txt
U  myproj/baz.c
U  myproj/qux.c
Updated to revision 31.
```

Puede cambiar parte de su copia local de trabajo a una rama si no desea cambiar toda su copia local de trabajo.

Si el lugar de su repositorio cambia y tiene una copia local de trabajo existente que querría continuar usando, puede usar **svn switch --relocate** para cambiar la URL de su copia local a otra:

```
$ svn checkout file:///tmp/repos test
A  test/a
A  test/b
…

$ mv repos newlocation
$ cd test/

$ svn update
svn: Unable to open an ra_local session to URL
svn: Unable to open repository 'file:///tmp/repos'

$ svn switch --relocate file:///tmp/repos file:///tmp/newlocation .
$ svn update
At revision 3.
```

Nombre

svn update — Actualiza su copia local de trabajo.

Sinopsis

```
svn update [PATH...]
```

Descripción

svn update trae cambios del repositorio a su copia local de trabajo. Si no se indica revisión alguna, actualizará su copia local de trabajo con la revisión `HEAD`. En caso contrario, sincronizará la copia local a la revisión indicada por el parámetro `--revision`.

Por cada elemento actualizado se mostrará una línea con un carácter representativo de la acción tomada. Estos indicadores tienen el siguiente significado:

A

Añadido

D

Borrado

U

Actualizado

C

En conflicto

G

Fusionado

Un carácter en la primera columna significa una actualización del fichero, mientras que las actualizaciones de las propiedades se muestran en la segunda columna.

Nombres alternativos

up

Cambios

En copia local

Accede al repositorio

Si

Parámetros

```
--revision (-r) REV
```

```
--non-recursive (-N)
--quiet (-q)
--diff3-cmd CMD
--username USER
--password PASS
--no-auth-cache
--non-interactive
--config-dir DIR
```

Ejemplos

Obtener cambios del repositorio que ocurrieron desde su última actualización:

```
$ svn update
A  newdir/toggle.c
A  newdir/disclose.c
A  newdir/launch.c
D  newdir/README
Updated to revision 32.
```

También puede actualizar su copia local de trabajo a una revisión anterior (Subversion no tiene el concepto de ficheros “pegajosos” como CVS; vea Apéndice A, *Subversion para usuarios de CVS*):

```
$ svn update -r30
A  newdir/README
D  newdir/toggle.c
D  newdir/disclose.c
D  newdir/launch.c
U  foo.c
Updated to revision 30.
```

Si quiere examinar una versión antigua de un único fichero, quizás prefiera usar **svn cat**.

svnadmin

svnadmin es la herramienta administrativa para monitorizar y reparar su repositorio Subversion. Para más información detallada, vea “svnadmin”.

Dado que **svnadmin** trabaja vía acceso directo al repositorio (y por lo tanto sólo puede ser usado en la máquina que almacena el mismo), hace referencia al repositorio con una ruta, no una URL.

Parámetros de svnadmin

`--bdb-log-keep`

(Específico de la BD Berkeley) Desactivar el borrado automático de los ficheros de registro de la base de datos.

`--bdb-txn-nosync`

(Específico de la BD Berkeley) Desactivar fsync cuando se finalizan transacciones de la base de datos.

`--bypass-hooks`

Ignorar el sistema de ganchos del repositorio.

`--clean-logs`

Elimina ficheros de registros de la BD Berkeley que no son usados.

`--force-uuid`

Por defecto, cuando carga datos en un repositorio que ya contiene revisiones, **svnadmin** ignorará el `UUID` del flujo de volcado. Este parámetro hará que el `UUID` del repositorio sea ajustado al `UUID` del flujo.

`--ignore-uuid`

Por defecto, cuando carga datos en un repositorio vacío, **svnadmin** usará el `UUID` del flujo de volcado. Este parámetro hará que ese `UUID` sea ignorado.

`--incremental`

Volcar una revisión sólo como un fichero diferencial respecto a la revisión anterior, en lugar del texto completo habitual.

`--parent-dir` *`DIR`*

Cuando se carga un volcado, el directorio raíz será *`DIR`* en lugar de `/`.

`--revision` (`-r`) *`ARG`*

Especifica una revisión particular sobre la cual operar.

`--quiet`

No muestra el progreso normal—muestra sólo errores.

Subcomandos de svnadmin

Nombre

svnadmin create — Crea un repositorio nuevo vacío.

Sinopsis

```
svnadmin create REPOS_PATH
```

Descripción

Crea un repositorio nuevo vacía en la ruta proporcionada. Si el directorio proporcionado no existe, será creado.[1]

Parámetros

```
--bdb-txn-nosync
--bdb-log-keep
```

Ejemplos

Crear un nuevo repositorio es así de fácil:

```
$ svnadmin create /usr/local/svn/repos
```

[1] Recuerde, **svnadmin** solamente funciona con *rutas* locales, no *URL*s.

Nombre

svnadmin deltify — Deltifica rutas cambiadas en un rango de revisiones.

Sinopsis

```
svnadmin deltify [-r LOWER[:UPPER]] REPOS_PATH
```

Descripción

svnadmin deltify únicamente existe en 1.0.x debido a razones históricas. Este comando está deprecado y ya no es necesario.

Tuvo su origen en tiempos que Subversion ofrecía a los administradores un control más preciso sobre las estrategias de compresión en el repositorio. Esto resultó ser una gran complejidad para *muy* poca ganancia, y la “característica” fue deprecada.

Parámetros

```
--revision (-r)
--quiet
```

Nombre

svnadmin dump — Vuelca el contenido de un sistema de ficheros por la salida estándar.

Sinopsis

```
svnadmin dump REPOS_PATH [-r LOWER[:UPPER]] [--incremental]
```

Descripción

Vuelca el contenido de un sistema de ficheros por la salida estándar en un formato “dumpfile” portable, enviando las respuestas por la salida estándar de errores. Vuelca revisiones de *`LOWER`* a *`UPPER`*. Si no se indican revisiones, vuelca todos los árboles de revisiones. Si sólo indica *`LOWER`*, vuelca ese árbol de revisiones. Vea usos prácticos en “Migrando un repositorio”.

Parámetros

```
--revision (-r)
--incremental
--quiet
```

Ejemplos

Vuelca su repositorio entero:

```
$ svnadmin dump /usr/local/svn/repos
SVN-fs-dump-format-version: 1
Revision-number: 0
* Dumped revision 0.
Prop-content-length: 56
Content-length: 56
…
```

Vuelca de forma incremental una única transacción de su repositorio:

```
$ svnadmin dump /usr/local/svn/repos -r 21 --incremental
* Dumped revision 21.
SVN-fs-dump-format-version: 1
Revision-number: 21
Prop-content-length: 101
Content-length: 101
…
```

Nombre

svnadmin help

Sinopsis

```
svnadmin help [SUBCOMMAND...]
```

Descripción

Este subcomando es útil cuando está atrapado en una isla desierta sin conexión por red o ejemplar de este libro.

Nombres alternativos

?, h

Nombre

svnadmin hotcopy — Realiza una copia en caliente del repositorio.

Sinopsis

```
svnadmin hotcopy OLD_REPOS_PATH NEW_REPOS_PATH
```

Descripción

Este subcomando realiza una copia de seguridad completa “en caliente” de su repositorio, incluyendo todos los ganchos, ficheros de configuración, y por supuesto, ficheros de la base de datos. Si pasa el parámetro `--clean-logs`, **svnadmin** realizará una copia en caliente de su repositorio, y tras ella eliminará los ficheros de registro no usados de la base de datos Berkeley del repositorio original. Puede ejecutar este comando en cualquier momento y realizar una copia segura del repositorio, sin importar si hay otros procesos usándolo.

Parámetros

`--clean-logs`

Nombre

svnadmin list-dblogs — Pregunta a la base de datos Berkeley qué ficheros de registro existen para un repositorio svn determinado.

Sinopsis

```
svnadmin list-dblogs REPOS_PATH
```

Descripción

La base de datos de Berkeley crea ficheros de registro de todos los cambios realizados en el repositorio, lo que permite que se recupere en caso de catástrofe. A no ser que active `DB_LOGS_AUTOREMOVE`, los ficheros de registro se irán acumulando, a pesar de que la mayoría no sean usados y puedan ser eliminados para recuperar espacio libre en disco. Vea “Gestionando el espacio de almacenamiento” para más información.

Nombre

svnadmin list-unused-dblogs — Pregunta a la base de datos Berkeley qué ficheros de registro pueden ser borrados con seguridad.

Sinopsis

```
svnadmin list-unused-dblogs REPOS_PATH
```

Descripción

La base de datos de Berkeley crea ficheros de registro de todos los cambios realizados en el repositorio, lo que permite que se recupere en caso de catástrofe. A no ser que desactive `DB_LOGS_AUTOREMOVE`, los ficheros de registro se irán acumulando, a pesar de que la mayoría no sean usados y puedan ser eliminados para recuperar espacio libre en disco. Vea “Gestionando el espacio de almacenamiento” para más información.

Ejemplos

Eliminar todos los ficheros de registro no usados del repositorio:

```
$ svnadmin list-unused-dblogs /path/to/repos
/path/to/repos/log.0000000031
/path/to/repos/log.0000000032
/path/to/repos/log.0000000033

$ svnadmin list-unused-dblogs /path/to/repos | xargs rm
## disk space reclaimed!
```

Nombre

svnadmin load — Lee un flujo “con formato de volcado” de la entrada estándar.

Sinopsis

```
svnadmin load REPOS_PATH
```

Descripción

Lee un flujo “con formato de volcado” de la entrada estándar, guardando nuevas revisiones en el sistema de ficheros del repositorio. Envía información del progreso por la salida estándar.

Parámetros

```
--quiet (-q)
--ignore-uuid
--force-uuid
--parent-dir
```

Ejemplo

Esto muestra el comienzo de la recuperación de un repositorio a partir de un fichero de copia de seguridad (realizado, por supuesto, con **svn dump**):

```
$ svnadmin load /usr/local/svn/restored < repos-backup
<<< Started new txn, based on original revision 1
     * adding path : test ... done.
     * adding path : test/a ... done.
…
```

O si desea cargarlo en un subdirectorio:

```
$ svnadmin load --parent-dir new/subdir/for/project /usr/local/svn/restored <
repos-backup
<<< Started new txn, based on original revision 1
     * adding path : test ... done.
     * adding path : test/a ... done.
…
```

Nombre

svnadmin lstxns — Imprime los nombres de todas las transacciones en curso.

Sinopsis

```
svnadmin lstxns REPOS_PATH
```

Descripción

Imprime los nombres de todas las transacciones en curso. Vea “Limpieza del repositorio” para más información sobre cómo las transacciones en curso son creadas y qué debería hacer con ellas.

Ejemplos

Mostrar todas las transacciones en curso de un repositorio.

```
$ svnadmin lstxns /usr/local/svn/repos/
1w
1x
```

Nombre

svnadmin recover — Recupera cualquier estado perdido en el repositorio.

Sinopsis

```
svnadmin recover REPOS_PATH
```

Descripción

Ejecute este comando si obtiene un error indicando que su repositorio debe ser recuperado.

Ejemplos

Recupera un repositorio colgado:

```
$ svnadmin recover /usr/local/svn/repos/
Acquiring exclusive lock on repository db.
Recovery is running, please stand by...
Recovery completed.
The latest repos revision is 34.
```

Nombre

svnadmin rmtxns — Elimina transacciones de un repositorio.

Sinopsis

```
svnadmin rmtxns REPOS_PATH TXN_NAME...
```

Descripción

Elimina transacciones excepcionales de un repositorio. Esto está cubierto con detalle en “Limpieza del repositorio”.

Parámetros

```
--quiet (-q)
```

Ejemplos

Elimina transacciones con nombre:

```
$ svnadmin rmtxns /usr/local/svn/repos/ 1w 1x
```

Afortunadamente, la salida de svn lstxns funciona muy bien como la entrada de rmtxns:

```
$ svnadmin rmtxns /usr/local/svn/repos/  `svnadmin lstxns /usr/local/svn/repos/`
```

Lo cual eliminará todas las transacciones pendientes de su repositorio.

Nombre

svnadmin setlog — Cambia el mensaje de informe de cambios de una revisión.

Sinopsis

```
svnadmin setlog REPOS_PATH -r REVISION FILE
```

Descripción

Cambia el mensaje de informe de cambios de la revisión REVISION con el contenido de FILE.

Esto es similar a usar **svn propset --revprop** para modificar la propiedad `svn:log` de una revisión, con la excepción de que también puede usar la opción `--bypass-hooks` para evitar la ejecución de enganches previos o posteriores a la modificación, lo cual es útil si la modificación de propiedades de revisión no ha sido activada en el gancho pre-revprop-change.

Las propiedades de revisiones no están bajo control de versiones, por lo que este comando sobreescribirá de forma permanente el mensaje de informe de cambios anterior.

Parámetros

```
--revision (-r) ARG
--bypass-hooks
```

Ejemplos

Cambia el mensaje de informe de cambios de la revisión 19 al contenido del fichero `msg`:

```
$ svnadmin setlog /usr/local/svn/repos/ -r 19 msg
```

Nombre

svnadmin verify — Verifica datos almacenados en el repositorio.

Sinopsis

```
svnadmin verify REPOS_PATH
```

Descripción

Ejecute este comando si desea verificar la integridad de su repositorio. Esto básicamente recorrerá todas las revisiones del repositorio realizando un volcado de las mismas descartando su salida.

Ejemplos

Verifica un repositorio bloqueado:

```
$ svnadmin verify /usr/local/svn/repos/
* Verified revision 1729.
```

svnlook

svnlook es una utilidad de línea de comando para examinar diferentes aspectos de un repositorio Subversion. No realiza ningún cambio en el repositorio—sólo se usa para "mirar". **svnlook** se usa típicamente en los ganchos de repositorio, pero el administrador del mismo puede encontrarlo útil para realizar diagnósticos.

Dado que **svnlook** funciona mediante el acceso directo al repositorio (y por lo tanto sólo puede ser usado en la máquina que lo almacena), trabaja con rutas de ficheros, no con URLs.

Si no especifica revisión o transacción alguna, **svnlook** por defecto usa la revisión más joven (la más reciente) del repositorio.

Parámetros de svnlook

Los parámetros de **svnlook** son globales, igual que en **svn** y **svnadmin**; no obstante, la mayoría de los parámetros sólo son aplicables a un subcomando dato que la funcionalidad de **svnlook** es (intencionadamente) limitada.

`--no-diff-deleted`

Evita que **svnlook** imprima las diferencias para los ficheros borrados. El comportamiento por defecto cuando un fichero es borrado en una transacción/revisión es imprimir las mismas diferencias que vería si hubiese dejado el fichero habiendo eliminado todo su contenido.

`--revision (-r)`

Especifica un número de revisión particular que desee examinar.

`--transaction (-t)`

Especifica un identificador de transacción particular que desee examinar.

`--show-ids`

Muestras los identificativos de las revisiones de los nodos del sistema de ficheros para cada ruta en el árbol.

svnlook

Nombre

svnlook author — Muestra el autor.

Sinopsis

```
svnlook author REPOS_PATH
```

Descripción

Muestra el autor de una revisión o transacción en el repositorio.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

svnlook author es útil, pero no muy excitante:

```
$ svnlook author -r 40 /usr/local/svn/repos
sally
```

Nombre

svnlook cat — Muestra el contenido de un fichero.

Sinopsis

```
svnlook cat REPOS_PATH PATH_IN_REPOS
```

Descripción

Muestra el contenido de un fichero.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra el contenido de un fichero en la transacción `ax8`, ubicado en `/trunk/README`:

```
$ svnlook cat -t ax8 /usr/local/svn/repos /trunk/README

               Subversion, a version control system.
               =====================================

$LastChangedDate: 2003-07-17 10:45:25 -0500 (Thu, 17 Jul 2003) $

Contents:

     I. A FEW POINTERS
    II. DOCUMENTATION
   III. PARTICIPATING IN THE SUBVERSION COMMUNITY
…
```

Nombre

svnlook changed — Muestra qué rutas fueron modificadas.

Sinopsis

```
svnlook changed REPOS_PATH
```

Descripción

Muestra las rutas que fueron modificadas en una revisión o transacción particular, al igual que una letra de estado “estilo svn update” en la primera columna: `A` para elementos añadidos, `D` para borrados, y `U` para actualizados (modificados).

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra un listado de todos los ficheros modificados en la revisión 39 de un repositorio de pruebas:

```
$ svnlook changed -r 39 /usr/local/svn/repos
A   trunk/vendors/deli/
A   trunk/vendors/deli/chips.txt
A   trunk/vendors/deli/sandwich.txt
A   trunk/vendors/deli/pickle.txt
```

Nombre

svnlook date — Muestra la fecha.

Sinopsis

```
svnlook date REPOS_PATH
```

Descripción

Muestra la fecha de una revisión o transacción de un repositorio.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra la fecha de la revisión 40 de un repositorio de pruebas:

```
$ svnlook date -r 40 /tmp/repos/
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
```

Nombre

svnlook diff — Muestra las diferencias de ficheros y propiedades modificadas.

Sinopsis

```
svnlook diff REPOS_PATH
```

Descripción

Muestra diferencias al estilo GNU de ficheros y propiedades modificadas en un repositorio.

Parámetros

```
--revision (-r)
--transaction (-t)
--no-diff-deleted
```

Ejemplos

Esto muestra un fichero (vacío) recientemente añadido, un fichero borrado, y un fichero copiado:

```
$ svnlook diff -r 40 /usr/local/svn/repos/
Copied: egg.txt (from rev 39, trunk/vendors/deli/pickle.txt)

Added: trunk/vendors/deli/soda.txt
===================================================================================

Modified: trunk/vendors/deli/sandwich.txt
===================================================================================
--- trunk/vendors/deli/sandwich.txt (original)
+++ trunk/vendors/deli/sandwich.txt 2003-02-22 17:45:04.000000000 -0600
@@ -0,0 +1 @@
+Don't forget the mayo!

Modified: trunk/vendors/deli/logo.jpg
===================================================================================
(Binary files differ)

Deleted: trunk/vendors/deli/chips.txt
===================================================================================

Deleted: trunk/vendors/deli/pickle.txt
===================================================================================
```

Si un fichero tiene una propiedad `svn:mime-type` no textual , entonces las diferencias no se muestran de forma explícita.

Nombre

svnlook dirs-changed — Muestra los directorios cuyas propiedades fueron modificadas.

Sinopsis

```
svnlook dirs-changed REPOS_PATH
```

Descripción

Muestra los directorios cuyas propiedades o ficheros hijo fueron modificados.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra los directorios que fueron modificados en la revisión 40 de nuestro repositorio de pruebas:

```
$ svnlook dirs-changed -r 40 /usr/local/svn/repos
trunk/vendors/deli/
```

Nombre

svnlook help

Sinopsis

```
También svnlook -h y svnlook -?.
```

Descripción

Muestra el mensaje de ayuda de svnlook. Este comando, al igual que su hermano **svn help**, es también su amigo, a pesar de no llamarlo y que olvidó invitarlo a su última fiesta.

Nombres alternativos

?, h

Nombre

svnlook history — Muestra información sobre la historia de una ruta en el repositorio (o del directorio raíz si no especifica ruta alguna).

Sinopsis

```
svnlook history REPOS_PATH
            [PATH_IN_REPOS]
```

Descripción

Muestra información sobre la historia de una ruta en el repositorio (o del directorio raíz si no especifica ruta alguna).

Parámetros

```
--revision (-r)
--show-ids
```

Ejemplos

Esto muestra la historia de la ruta `/tags/1.0` hasta la revisión 20 en nuestro repositorio de pruebas.

```
$ svnlook history -r 20 /usr/local/svn/repos /tags/1.0 --show-ids
REVISION   PATH <ID>
--------   ---------
      19   /tags/1.0 <1.2.12>
      17   /branches/1.0-rc2 <1.1.10>
      16   /branches/1.0-rc2 <1.1.x>
      14   /trunk <1.0.q>
      13   /trunk <1.0.o>
      11   /trunk <1.0.k>
       9   /trunk <1.0.g>
       8   /trunk <1.0.e>
       7   /trunk <1.0.b>
       6   /trunk <1.0.9>
       5   /trunk <1.0.7>
       4   /trunk <1.0.6>
       2   /trunk <1.0.3>
       1   /trunk <1.0.2>
```

Nombre

svnlook info — Muestra el autor, la fecha, el mensaje de informe de cambios y su tamaño.

Sinopsis

```
svnlook info REPOS_PATH
```

Descripción

Muestra el autor, la fecha, el mensaje de informe de cambios y su tamaño.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra la información sobre la revisión 40 en nuestro repositorio de pruebas.

```
$ svnlook info -r 40 /usr/local/svn/repos
sally
2003-02-22 17:44:49 -0600 (Sat, 22 Feb 2003)
15
Rearrange lunch.
```

Nombre

svnlook log — Muestra el mensaje de informe de cambios.

Sinopsis

```
svnlook log REPOS_PATH
```

Descripción

Muestra el mensaje de informe de cambios.

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra el mensaje de informe de cambios para la revisión 40 de nuestro repositorio de pruebas:

```
$ svnlook log /tmp/repos/
Rearrange lunch.
```

Nombre

svnlook propget — Muestra el valor de una propiedad en una ruta del repositorio.

Sinopsis

```
svnlook propget REPOS_PATH PROPNAME PATH_IN_REPOS
```

Descripción

Muestra el valor de una propiedad en una ruta del repositorio.

Nombres alternativos

pg, pget

Parámetros

```
--revision (-r)
--transaction (-t)
```

Ejemplos

Esto muestra el valor de la propiedad “seasonings” del fichero `/trunk/sandwich` en la revisión `HEAD`:

```
$ svnlook pg /usr/local/svn/repos seasonings /trunk/sandwich
mustard
```

Nombre

svnlook proplist — Muestra los nombres y valores de propiedades versionadas de ficheros y directorios.

Sinopsis

```
svnlook proplist REPOS_PATH PATH_IN_REPOS
```

Descripción

Muestra las propiedades de una ruta en el repositorio. Con `--verbose`, también muestra los valores de las propiedades.

Nombres alternativos

pl, plist

Parámetros

```
--revision (-r)
--transaction (-t)
--verbose (-v)
```

Ejemplos

Esto muestra los nombres de las propiedades asociadas al fichero `/trunk/README` en la revisión `HEAD`:

```
$ svnlook proplist /usr/local/svn/repos /trunk/README
  original-author
  svn:mime-type
```

Esto es el mismo comando que en el ejemplo anterior, pero esta vez también se muestran los valores de las propiedades:

```
$ svnlook proplist /usr/local/svn/repos /trunk/README
  original-author : fitz
  svn:mime-type : text/plain
```

Nombre

svnlook tree — Muestra un árbol de directorios.

Sinopsis

```
svnlook tree REPOS_PATH [PATH_IN_REPOS]
```

Descripción

Muestra un árbol de directorios, comenzando en *`PATH_IN_REPOS`* (si fue proporcionado, en caso contrario en la raíz), opcionalmente enseñando los identificadores de las revisiones de los nodos.

Parámetros

```
--revision (-r)
--transaction (-t)
--show-ids
```

Ejemplos

Esto muestra el árbol de directorios (con IDs de nodo) de la revisión 40 en nuestro repositorio de pruebas:

```
$ svnlook tree -r 40 /usr/local/svn/repos --show-ids
/ <0.0.2j>
 trunk/ <p.0.2j>
  vendors/ <q.0.2j>
   deli/ <1g.0.2j>
    egg.txt <1i.e.2j>
    soda.txt <1k.0.2j>
    sandwich.txt <1j.0.2j>
```

Nombre

svnlook uuid — Muestra el `UUID` del repositorio.

Sinopsis

```
svnlook uuid REPOS_PATH
```

Descripción

Muestra el `UUID` del repositorio. El UUID es el *ID*entificador *U*niversal *U*nico. El cliente de Subversion usa este identificador para diferenciar un repositorio de otro.

Ejemplos

```
$ svnlook uuid /usr/local/svn/repos
e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
```

Nombre

svnlook youngest — Muestra el número de revisión más joven.

Sinopsis

```
svnlook youngest REPOS_PATH
```

Descripción

Muestra el número de revisión más joven de un repositorio.

Ejemplos

Esto muestra la revisión más joven de nuestro directorio de pruebas:

```
$ svnlook youngest /tmp/repos/
42
```

svnserve

svnserve permite el acceso a repositorios Subversion usando el protocolo de red `svn`. Puede ejecutar svnserve como proceso servidor independiente, o puede tener otro proceso, como **inetd**, **xinetd** o **sshd**, que lo lance para usted.

Una vez el cliente selecciona un repositorio transmitiendo su URL, **svnserve** lee el fichero `conf/svnserve.conf` en el directorio del repositorio para determinar los parámetros específicos de repositorio, como qué base de datos de autenticación usar y qué políticas de autorización aplicar. Vea “svnserve, un servidor personalizado” para más detalles sobre el fichero `svnserve.conf`.

Parámetros de svnserve

A diferencia de los comandos que hemos descrito previamente, **svnserve** no tiene sub comandos—**svnserve** se controla exclusivamente mediante parámetros.

`--daemon` (`-d`)

Hace que **svnserve** se ejecute en modo demonio. **svnserve** se lanza a sí mismo segundo plano y acepta y sirve conexiones TCP/IP en el puerto svn (3690, por defecto).

`--listen-port=`*`PORT`*

Hace que **svnserve** escuche en el puerto *`PORT`* cuando se ejecuta en modo demonio.

`--listen-host=`*`HOST`*

Hace que **svnserve** escuche en *`HOST`*, que puede ser un nombre de máquina o una dirección IP.

`--foreground`

Usado junto con `-d`, este parámetro hace que **svnserve** se quede en primer plano. Este parámetro es útil principalmente para depurar.

`--inetd` (`-i`)

Hace que **svnserve** use los descriptores de fichero stdin/stdout, tal y como es apropiado para un demonio que se ejecuta desde **inetd**.

`--help` (`-h`)

Muestra un resumen de uso y finaliza la ejecución.

`--root=`*`ROOT`* (`-r=`*`ROOT`*)

Establece la raíz virtual para repositorios servidos por **svnserve**. La ruta en las URLs proporcionadas por el cliente serán interpretadas como relativas a esta raíz, y no se les permitirá escapar de ahí.

`--tunnel` (`-t`)

Hace que **svnserve** se ejecute en modo túnel, que es igual que el modo de operación **inetd** (sirviendo una conexión por stdin/stdout) con la excepción de que la conexión se considerada preautenticada con el nombre de usuario del uid actual. Este parámetro es seleccionado por el cliente cuando se ejecuta con un agente de túneles como **ssh**.

`--threads` (`-T`)

Cuando se usa en modo demonio, hace que **svnserve** lance un hilo de ejecución en lugar de un proceso por cada conexión. El proceso **svnserve** seguirá lanzándose en segundo plano al ser iniciado.

`--listen-once` (`-X`)

Hace que **svnserve** acepte una conexión en el puerto svn, la sirva, y termine. Esta opción es principalmente útil para depurar.

Apéndice A. Subversion para usuarios de CVS

Este apéndice es una guía para usuarios de CVS nuevos en Subversion. Es esencial una lista de diferencias entre los dos sistemas "visto desde 10,000 pies de alto". Para cada sección, proporcionaremos cuando sea posible enlaces a los capítulos relevantes.

Aunque el fin de Subversion es asumir el control de la base actual y futura de los usuarios de CVS, algunas nuevas características y cambios de diseño fueron requeridos para arreglar ciertos comportamientos "rotos" que tenía CVS. Esto significa que, como usuario de CVS, puede necesitar romper hábitos—unos que usted olvidó....

Los números de revisión son diferentes ahora

En CVS, los números de revisión son por-fichero. Esto es porque CVS usa RCS como motor de programa; cada fichero tiene su correspondiente fichero RCS en el repositorio, y el repositorio se representa aproximadamente según la estructura de su árbol de proyecto.

En Subversion, el repositorio parece un solo sistema de ficheros. Cada envío tiene como resultadoun nuevo árbol de sistema de ficheros entero; en esencia, el repositorio es un array de árboles. Cada uno de estos árboles se etiqueta con un solo número de revisión. Cuando alguien habla sobre la "revisión 54", ellos están hablando sobre un árbol particular (e indirectamente, la forma que el sistema de ficheros tenía después del envío 54).

Técnicamente, no es válido hablar sobre la "revisión 5 de `foo.c`". En su lugar, uno diría "`foo.c` como aparece en la revisión 5". También, tenga cuidado cuando haga supuestos sobre la evolución de un fichero. En CVS, las revisiones 5 y 6 de `foo.c` son siempre diferentes. En Subversion, es más probable que `foo.c` *no* haya cambiado entre las revisiones 5 y 6.

Para más detalles sobre este asunto, vea "Revisiones".

Versiones de directorios

Subversion sigue estructura de árbol, no solo el contenido de ficheros. Es una de las razones más grandes por las que Subversion fue escrito para reemplazar CVS.

Esto es lo que significa para usted, como anterior usuario de CVS:

- Los comandos **svn add** y **svn delete** trabajan ahora sobre directorios, tal como trabajan sobre ficheros. Así que haga **svn copy** y **svn move**. Sin embargo, estos comandos *no* causan ningún tipo de cambio inmediato en el repositorio. En su lugar, los objetos de la copia de trabajo simplemente se planifican para la adición o la eliminación. No sucede ningún cambio en el repositorio hasta que ejecute **svn commit**.

- Los directorios no serán contenedores estúpidos nunca más; tienen número de revisión como los ficheros. (O más correctamente, es correcto hablar sobre "directorio `foo/` en la revisión 5".)

Vamos a hablar más sobre este último punto. El versionado de directorios es un duro problema; porque queremos permitir copias de trabajo con revisiones mezcladas, hay algunas limitaciones en cómo podemosabusar de este modelo.

Desde un punto de vista teórico, definimos "revisión 5 del directorio `foo`" para significar una colección específica de directorios-entradas y características. Ahora suponga que empezamos a añadir y a quitar ficheros de `foo`, y luego enviamos el cambio. Sería una mentira decir que todavía tenemos la revisión 5 de `foo`. Sin embargo, si vemos el número de revisión de `foo` después del envío, también sería esto una mentira; puede haber otros cambios en `foo` que todavía no hayamos recibido, porque no hemos actualizado todavía.

Subversion trata este problema siguiendo silenciosamente las agregaciones y las eliminaciones enviadas en el área `.svn`. Cuando usted ejecuta finalmente **svn update**, todas las cuentas se fijan con el repositorio, y el nuevo número de revisión se fija correctamente. *Por lo tanto, solamente después de una actualización es verdaderamente seguro decir que usted tiene una revisión "perfecta" de un directorio.* La mayoría del tiempo, su copia de trabajo contendrá revisiones "imperfectas" del directorio.

Similarmente, un problema surge si usted intenta enviar cambios de las características de un directorio. Normalmente, el envío chocaría con el número local de revisión del directorio de trabajo. Pero otra vez, eso sería una mentira, porque puede haber añadidos o eliminaciones que el directorio todavía no tiene, porque no ha habido ninguna actualización. *Por lo tanto, usted no tiene permiso para enviar cambios-características sobre un directorio a menos que el directorio esté actualizado.*

Para más discusión sobre las limitaciones del versionado de directorios, vea "Las limitaciones de las revisiones mixtas".

Más operaciones estando desconectado

En años recientes, ha llegado a ser muy barata y abundante, pero el ancho de banda de red no. Por lo tanto, la copia de trabajo se Subversion ha sido optimizada alrededor del recurso más escaso.

El directorio administrativo `.svn` tiene el mismo propósito que el directorio `CVS`, excepto que este también almacena copias "prístinas" de solo-lectura de sus ficheros. Esto le permite hacer muchas cosas desconectado:

svn status

Le muestra cualquier cambio local que haya hecho (vea "**svn status**")

svn diff

Le muestra los detalles de sus cambios (vea "**svn diff**")

svn revert

Borra sus cambios locales (vea "**svn revert**")

También, los ficheros prístinos almacenados permiten al cliente de Subversion enviar las diferencias al repositorio, lo que CVS no puede hacer.

El último subcomando en la lista es nuevo; este no solo borrará las modificaciones locales, sino que desprogramará las operaciones tales como adiciones y eliminaciones. Es la manera preferida para revertir un fichero; ejecutando **rm file; svn update** seguirá funcionando, pero empaña el propósito de actualización. Y, mientras nosotros estamos en este asunto…

Distinciones entre estado (status) y actualización (update)

En Subversion, hemos intentado eliminar muchas de las confusiones entre los comandos **cvs status** y **cvs update**.

El comando **cvs status** tiene dos propósitos: primero, mostrarle al usuario cualquier modificación local en la copia de trabajo, y segundo, mostrarle al usuario qué ficheros están desactualizados. Desafortunadamente, debido a la difícil de leer salida de estado, muchos usuarios de CVS no se aprovechan del todo de este comando. En cambio, han desarrollado el habito de ejecutar **cvs update** o **cvs update -n** para ver rápidamente sus modificaciones. Si los usuarios olvidan usar la opción `-n`, esto tiene el efecto secundario de combinar cambios del repositorio que no están preparados.

Con Subversion, hemos intentado eliminar este desorden haciendo la salida de **svn status** fácil de leer para los humanos y los programas de análisis sintácticos. También, **svn update** solo imprime información sobre los ficheros que están actualizados, *no* sobre modificaciones locales.

svn status imprime todos los ficheros que tienen modificaciones locales. Por defecto, no se contacta con el repositorio. Mientras este subcomando acepta un número justo de opciones, los siguientes son los usados más comúnmente:

`-u`

Contacta con el repositorio para determinar, y después mostrar, la información más desactualizada.

`-v`

Muestra *todas* las entradas bajo el control de versiones.

`-N`

Se ejecuta no-recursivamente (no desciende hacia los subdirectorios).

El comando **status** tiene dos formatos de salida. En el formato “corto” por defecto, las modificaciones locales se parecen a esto:

```
% svn status
M     ./foo.c
M     ./bar/baz.c
```

Si usted especifica la opción `--show-updates` (`-u`), un formato de salida más largo es usado:

```
% svn status -u
M             1047    ./foo.c
       *      1045    ./faces.html
       *         -    ./bloo.png
M             1050    ./bar/baz.c
Status against revision:   1066
```

En este caso, aparecen dos columnas nuevas. La segunda columna contiene un asterisco si el fichero o directorio está desactualizado. La tercera columna muestra el número de revisión del objeto en la copia de trabajo. En el ejemplo de arriba, el asterisco indica que `faces.html` será parcheado si actualizamos, y que `bloo.png` es un nuevo fichero añadido en el repositorio. (El `-` al lado de `bloo.png` significa que todavía no existe en la copia de trabajo.)

Por último, aquí está un resumen rápido de los códigos de estado más comunes que usted puede ver:

```
A    Resource is scheduled for Addition
D    Resource is scheduled for Deletion
M    Resource has local modifications
C    Resource has conflicts (changes have not been completely merged
       between the repository and working copy version)
X    Resource is external to this working copy (comes from another
       repository.  See “svn:externals”)
?    Resource is not under version control
!    Resource is missing or incomplete (removed by another tool than
       Subversion)
```

Subversion ha combinado los códigos de CVS **P** y **U** en **U**. Cuando ocurre una fusión o un conflicto, Subversion simplemente imprime **G** o **C**, antes que una oración entera sobre esto.

Para una discusión más detallada sobre **svn status**, vea “**svn status**”.

Ramas y etiquetas

Subversion no distingue entre el espacio del sistema de ficheros y el espacio de “rama”; las ramas y etiquetas son directorios ordinarios dentro del sistema de ficheros. Ésta es probablemente la única gran valla mental que un usuario de CVS tiene que superar. Lea todo sobre esto en Capítulo 4, *Crear ramas y fusionarlas*.

Puesto que Subversion trata las ramas y etiquetas como directorios ordinarios, recuerde siempre descargar el directorio `trunk` (`http://svn.example.com/repos/calc/trunk/`) de su proyecto, y no el proyecto en si mismo (`http://svn.example.com/repos/calc/`). Si comete el error de descargar el proyecto en sí mismo, obtendrá una copia de trabajo que contendrá una copia de su proyecto para cada rama y etiqueta que tenga.[1]

Propiedades de los metadatos

Una característica nueva de Subversion es que usted puede unir metadatos arbitrarios (o “propiedades”) a ficheros y directorios. Las propiedades son pares arbitrarios nombre/valor asociados con ficheros y directorios en su copia de trabajo.

Para fijar u obtener un nombre de propiedad, use los subcomandos **svn propset** y **svn propget**. Para enumerar todas las propiedades de un objeto, use **svn proplist**.

Para más información, vea “Propiedades”.

Resolución de conflictos

CVS marca conflictos en linea con “marcas de conflicto”, e imprime una **C** durante la actualización. Históricamente, esto ha causado problemas, porque CVS no está haciendo bastante. Muchos usuarios se olvidan (o no ven) la **C** después de que esta silbe en sus terminales. Se olvidan a menudo de que las marcas de conflicto están incluso presentes, y después accidentalmente envían los ficheros conteniendo las marcas de conflicto.

Subversion solventa este problema haciendo los conflictos más tangibles. Recuerde que un fichero está en estado de conflicto, y no permitirá que envíe sus cambios hasta que ejecute **svn resolved**. Vea “Resolver conflictos (fusionando los cambios de otros)” para más detalles.

Ficheros binarios y traducción

En el sentido más general, Subversion maneja ficheros binarios más elegantemente que CVS. Porque CVS usa RCS, solo puede almacenar copias completas sucesivas de un fichero binario que cambia. Pero internamente, Subversion expresa diferencias entre ficheros usando un algoritmo de diferenciación de binarios, sin importar si contienen datos o datos binarios. Esto significa que todos los ficheros son almacenados diferenciados (comprimidos) en el repositorio, y las pequeñas diferencias son siempre enviadas a través de la red.

Los usuarios de CVS tienen que marcar los ficheros binarios con los parámetros `-kb`, para evitar que los datos sean mutilados (debido a la traducción de la expansión de la palabra clave y del final de linea. A veces se olvidan de hacerlo.

Subversion toma la ruta más paranoica: primero, nunca realiza ningún tipo de traducción de la palabra clave o del fin de linea a menos que usted le diga explícitamente que lo haga (vea “`svn:keywords`” y “`svn:eol-style`” para más detalles). Por defecto, Subversion trata todos los datos del ficheros como una cadena literal de bytes, y los ficheros son siempre guardados en el repositorio en un estado sin traducir.

Segundo, Subversion mantiene una noción interna de si un fichero es “texto” o datos “binarios”, pero esta noción es *solamente* existente en la copia de trabajo. Durante un **svn update**, Subversion realizará fusiones contextuales en ficheros de texto modificados localmente, pero no intentará hacer eso para los ficheros binarios.

[1] Esto es, proporcionando no ejecute hacia afuera el espacio en disco antes de que su descarga finalice.

Para determinar si es posible una fusión de contexto, Subversion examina la propiedad `svn:mime-type`. Si el fichero no tiene la propiedad `svn:mime-type`, o tiene un tipo MIME que es textual (por ejemplo `text/*`), Subversion asume que es texto. Si no, Subversion asume que el fichero es binario. Subversion también ayuda a los usuarios ejecutando un algoritmo de detección de binarios en los comandos **svn import** y **svn add**. Estos comandos harán una buena conjetura y después (posiblemente) fije una propiedad `svn:mime-type` binaria en el fichero que es agregado. (Si Subversion supone mal, el usuario siempre puede borrar o editar a mano la propiedad.)

Versionado de módulos

A diferencia de CVS, una copia de trabajo de Subversion es consciente que ha descargado un modulo. Eso significa que si alguien cambia la definición de un módulo, entonces una llamada a **svn update** actualizará la copia de trabajo apropiadamente.

Subversion define los módulos como una lista de directorios dentro de una propiedad del directorio: vea “Repositorios externos”.

Autenticación

Con el servidor de CVS, usted está obligado a “loguearse” al servidor antes de cualquier operación de lectura o escritura—usted incluso tiene que loguearse para operaciones anónimas. Con un repositorio de Subversion usando Apache HTTPD como el servidor, en principio usted no proporcionará ningún credencial de autenticación—si una operación que usted realiza requiere de autenticación, el servidor le desafiará para sus credenciales (donde esos credenciales son username y password, una certificación de cliente, o incluso ambos). De este modo si su repositorio es world-readable no le requerirán autenticarse para las operaciones de lectura.

Igual que con CVS, Subversion guarda una copia de sus credenciales en disco (en su directorio `~/.subversion/auth/`) a no ser que le indique que no lo haga usando el parámetro `--no-auth-cache`.

Convirtiendo un repositorio de CVS a Subversion

Quizá la manera más importante de familiarizar a usuarios de CVS con Subversion es dejarles continuar trabajando en sus proyectos usando el nuevo sistema. Y mientras que eso se puede lograr usando una importación plana a un repositorio de Subversion de un repositorio exportado de CVS, la solución más cuidadosa implica el transferir no solo la última foto de sus datos, sino toda la historia detrás de esta también, a partir de un sistema al otro. Éste es un problema extremadamente difícil a solucionar que implica la deducción de cambios en ausencia de atomicidad, y traducción entre las políticas de ramificación totalmente ortogonal de los sistemas, entre otras complicaciones. No obstante, hay un puñado de herramientas que que demandan apoyar por lo menos parcialmente la capacidad de convertir los repositorios existentes de CVS en unos de Subversion.

Una herramienta de estas es cvs2svn (`http://cvs2svn.tigris.org/`), un script de Python creado originalmente por miembros miembros de la propia comunidad de desarrollo de Subversion. Otros incluidos el módulo conversor para la herramienta VCP de Chia-liang Kao (`http://svn.clkao.org/revml/branches/svn-perl/`) y el RefineCVS de Lev Serebryakov's (`http://lev.serebryakov.spb.ru/refinecvs/`). Estas herramientas tienen varios niveles en estado completo, y pueden tomar decisiones enteramente diferentes sobre cómo manejar la historia de su repositorio de CVS. Cualquier herramienta que decida usar, asegúrese de realizar tantas verificaciones como pueda basándose en el resultado de conversión—¡después de todo, usted ha trabajado duro para construir esa historia!

Para una colección actualizada de enlaces a herramientas conversoras conocidas, visite la página de enlaces del website de Subversion (`http://subversion.tigris.org/project_links.html`).

Apéndice B. Solución de problemas

Problemas comunes

Hay un número de problemas que usted puede encontrar en el transcurso de instalar y usar Subversion. Algunos de estos serán resueltos una vez que usted tenga una mejor idea de cómo hace las cosas Subversion, mientras que otros son causados porque usted está acostumbrado a la manera de funcionar de otros sistemas de control de versiones. Otros problemas pueden ser insalvables debido a fallos en alguno de los sistemas operativos donde Subversion funciona (considerando la amplia gama de SO donde Subversion funciona, es asombroso que no encontramos demasiados de éstos).

La siguiente lista ha sido completada en el transcurso de años de usar Subversion. Si no puede encontrar aquí el problema que está teniendo, mire la versión más actualizada del FAQ en el página web principal de Subversion. Si todavía está atascado, entonces envíe un email a `<users@subversion.tigris.org>` con una descripción detallada del problema que está teniendo. [1]

Problemas usando Subversion

Aquí hay algunas de las preguntas más populares del FAQ de Subversion.

Cada vez que intento acceder a mi repositorio, mi cliente de Subversion se cuelga

Su repositorio no está corrupto, ni sus datos perdidos. Si su proceso accede directamente al repositorio (mod_dav_svn, svnlook, svnadmin, o si accede a una URL `file://`), entonces está usando Berkeley DB para acceder a sus datos. Berkeley DB es un sistema transaccional, lo que significa que registra todo sobre lo que va a hacer antes de hacerlo. Si su proceso es interrumpido (por ejemplo por una señal kill o segfault), entonces un fichero de bloqueo se deja atrás, junto con un fichero de registro que describe el trabajo inacabado. Cualquier otro proceso que procure tener acceso a la base de datos se colgará, esperando a que desaparezca el fichero de bloqueo. Para despertar su repositorio, necesita pedirle a Berkeley DB que finalice el trabajo, o rebobinar la base de datos a un estado anterior que se sepa que es consistente.

Asegúrese que ejecuta este comando como el usuario que posee y administra la base de datos, no como superusuario, o sino dejará ficheros pertenecientes al superusuarioen el directorio db. Estos ficheros no pueden ser abiertos por el usuario no root que administra la base de datos, que típicamente es usted o su proceso Apache. También asegúrese de tener fijada correctamente la umask cuando ejecute la recuperación, si falla esta bloqueará a usuarios que están en el grupo permitido para acceder al repositorio.

Simplemente ejecute:

```
$ svnadmin recover /path/to/repos
```

Una vez el comando se haya completado, compruebe los permisos en el directorio `db/` del repositorio.

Cada vez que intento ejecutar svn, dice que mi copia de trabajo está bloqueada.

La copia de trabajo de Subversion, como el Berkeley DB, usa un mecanismo transaccional para realizar todas las acciones. Esto es, registra todo acerca de lo que va a hacer antes de hacerlo. Si **svn** es interrumpido mientras realiza una acción, entonces uno o más ficheros de bloqueo son dejados atrás, junto con ficheros de registro describiendo las acciones inacabadas. (**svn status**mostrará una `L` al lado de los directorios bloqueados.)

Cualquier otro proceso que intente tener acceso a la copia de trabajo fallará cuando vea los bloqueos. Para despertar su copia de

[1] Recuerde que la cantidad de detalle que provea sobre su configuración y su problema es directamente proporcional a la probabilidad de conseguir una respuesta de la lista de correo. Está animado a incluir brevemente lo que ha tomado para desayunar o el nombre de soltera de su madre.

trabajo, usted necesita pedirle al cliente que finalice el trabajo. Para corregir esto, ejecute este comando desde lo alto de su copia de trabajo:

```
$ svn cleanup
```

Estoy teniendo errores encontrando o abriendo un repositorio, pero sé que mi URL del repositorio es correcta.

Vea “Cada vez que intento acceder a mi repositorio, mi cliente de Subversion se cuelga”.

También puede tener un problema de permisos al abrir el repositorio. Vea “Ofrecer múltiples métodos de acceso al repositorio”.

Cómo puedo especificar una letra de un dispositivo Windows en una URL `file://`?

Vea URLs del repositorio.

Estoy teniendo problemas haciendo operaciones de escritura en un repositorio de Subversion sobre una red.

Si la importación funciona bien sobre acceso local:

```
$ mkdir test
$ touch test/testfile
$ svn import test file:///var/svn/test -m "Initial import"
Adding         test/testfile
Transmitting file data .
Committed revision 1.
```

Pero no desde un puesto remoto:

```
$ svn import test http://svn.red-bean.com/test -m "Initial import"
harry's password: xxxxxxx

svn_error: … The specified activity does not exist.
```

Hemos visto esto cuando el directorio `REPOS/dav/` no tiene permisos de escritura para el proceso httpd. Compruebe los permisos para asegurarse que el httpd de Apache puede escribir en el directorio `dav/` (y al directorio correspondiente `db/`, por supuesto).

Bajo Windows XP, a veces el servidor de Subversion parece enviar datos corruptos.

Necesita instalar Windows XP Service Pack 1 para corregir un fallo en la pila TCP/IP en el sistema operativo. Puede conseguir toda clase de información sobre este Service Pack en `http://support.microsoft.com/default.aspx?scid=kb;EN-US;q317949`.

¿Cual es el mejor método de hacer un trazo de red de la conversación entre un cliente de Subversion y un servidor Apache?

Utilice Ethereal para cuchichearla conversación:

Las siguientes instrucciones son especificas para la versión gráfica de Ethereal, y puede no ser aplicable a la versión de linea de comandos (cuyo binario se nombra generalmente **tethereal**).

- Desplegar el menú Capture, y elija Start.
- Escriba puerto 80 para Filter, y desactive el modo promiscuo.
- Ejecute su cliente Subversion.
- Pulse Stop. Ahora usted tiene una captura. Parece una lista enorme de lineas.
- Haga clic en la columna Protocol para abreviar.
- Después, haga clic en la primera linea relevante TCP para seleccionarla.
- Clic derecho, y elija "Follow TCP Stream". Le presentarán los pares petición/respuesta de la conversión del HTTP del cliente de Subversion.

Alternativamente, usted puede fijar un parámetro en el fichero `servers` de configuración de tiempo de ejecución de su cliente para hacer aparecer la salida depurada. El valor numérico de neon-debug es una combinación de los valores `NE_DBG_*` en el fichero de cabecera `ne_utils.h`. Fijando la variable `neon-debug-mask` a 130 (por ejemplo `NE_DBG_HTTP + NE_DBG_HTTPBODY`) hará que los datos de HTTP sean mostrados.

Usted puede querer inhabilitar la compresión al hacer una traza de red arreglando el parámetro `http-compression` en el mismo fichero.

Acabo de construir la distribución binaria, y cuando intento descargar Subversion, consigo un error sobre un "Esquema desconocido de URL."

Subversion usa un sistema de plugins para permitir el acceso a los repositorios.Actualmente hay tres de estos plugins: ra_local permite el acceso a un repositorio local, ra_dav permite el acceso a un repositorio vía WebDAV, y ra_svn permite el acceso local o remoto vía el servidor svnserve. Cuando intente realizar una operación en subversion, el programa intenta cargar dinámicamente un plugin basado en el esquema de URL. Una URL `file://` intentará cargar ra_local, y una URL `http://` intentará cargar ra_dav.

El error que usted está viendo significa que el enlazador/cargador dinámico no puede encontrar los plugins a cargar. Esto normalmente ocurre cuando compila subversion con librerías compartidas, entonces intentar ejecutarlo sin ejecutar primero 'make install'. Otra causa posible es que usted ejecutó make install, pero las librerías fueron instaladas en una ubicación que el enlazador/cargador dinámico no reconoce. Bajo Linux, usted puede permitir al enlazador/cargador buscar las librerías añadiendo el directorio de biblioteca a `/etc/ld.so.conf` y ejecutando ldconfig. Si no desea hacer esto, o no tiene acceso de root, usted también puede especificar el directorio de librería en la variable de entorno LD_LIBRARY_PATH.

¿Por qué el comando 'svn revert' requiere un target explícito? ¿Por qué no es recursivo por defecto? Este comportamiento difiere de casi el resto de subcomandos.

La respuesta corta: es para su propio bien.

Subversion pone una prioridad muy alta en la protección de sus datos, y no solo en sus datos versionados. Las modificaciones que usted hace a los ficheros recién versionados, y los nuevos ficheros programados para ser añadidos al sistema de control de versiones, deben ser tratados con cuidado.

Haciendo el comando **svn revert** requiere un objetivo explícito—incluso si este objetivo es '.'—es una manera de lograr eso. Este requisito (así como requerirle para proveer el parámetro `--recursive` si usted desea este comportamiento) está pensado para hacerle pensar realmente que está haciendo, porque una vez que se invierten sus ficheros, sus modificaciones locales se van para siempre.

Cuando arranco Apache, mod_dav_svn se queja por una "mala versión de la base de datos", que encontró db-3.X, en vez de db-4.X.

Su apr-util está enlazado contra DB-3, y svn enlazado contra DB-4. Desafortunadamente, los símbolos de la BD no son diferentes. Cuando mod_dav_svn está cargado en el espacio de proceso de Apache, termina resolviendo los nombres de los símbolos contra la librería DB-3 de apr-util.

La solución es asegurarse de que apr-util compila contra DB-4. Puede hacer esto pasando opciones específicas al fichero de configuración de apr-util o al de apache: "--with-dbm=db4 --with-berkeley-db=/the/db/prefix".

Estoy obteniendo errores de "Función no implementada" en RedHat 9, y nada funciona. ¿Cómo arreglo esto?

Esto realmente no es un problema con Subversion, pero afecta a menudo a usuarios de Subversion.

RedHat 9 y Fedora vienen con una librería de Berkeley DB que confía en el soporte del núcleo para NPTL (la Librería Posix Nativa de Threads). Los núcleos que proporciona RedHat tiene este soporte incluido, pero si usted compila su propio núcleo, entonces puede no tener el soporte para NPTL. Si este es el caso, entonces usted verá errores como estos:

```
svn: Berkeley DB error
svn: Berkeley DB error while creating environment for filesystem tester/db:
Function not implemented
```

Esto puede ser arreglado en una de varias maneras:

- Reconstruir db4 para el núcleo que usted está usando.
- Usar un núcleo de RedHat 9.
- Aplicar los parches NPTL al núcleo que está usando.
- Usar un núcleo reciente (2.5.x) con soporte NPTL incluido.
- Comprobar si la variable de entorno `LD_ASSUME_KERNEL` está definida como 2.2.5, y si es así, desasignarla antes de arrancar Subversion (Apache). (Generalmente usted fijaría esta variable para ejecutar Wine o Winex en RedHat 9)

¿Por qué el informe de cambios dice "(no author)" para los ficheros envia-

dos o importados vía Apache (ra_dav)?

Si usted permite acceso anónimo de escritura al repositorio vía Apache, el servidor Apache nunca pide al cliente un nombre de usuario, y en lugar de eso permite la operación de escritura sin autenticación. Dado que Subversion no tiene ni idea de quien hizo la operación, esto da lugar a un registro como este:

```
$ svn log
------------------------------------------------------------------------
rev 24:  (no author) | 2003-07-29 19:28:35 +0200 (Tue, 29 Jul 2003)
…
```

Lea sobre añadir autenticación en Capítulo 6, *Configuración del servidor*.

Estoy obteniendo errores ocasionales de "Acceso Denegado" en Windows. Parece que sucede al azar.

Éstos aparecen debido a varios servicios de Windows que supervisan el sistema de ficheros en busca de cambios (software antivirus, servicios de indexado, el COM+ Event Notification Service). Esto realmente no es un bug en Subversion, lo que lo hace difícil de corregir. Un resumen del estado actual de investigación está disponible en `http://www.contactor.se/~dast/svn/archive-2003-10/0136.shtml`. Un trabajo que debe reducir la tasa de incidencias para la mayoría de la gente fue implementada en la revisión 7598.

En FreeBSD, ciertas operaciones (especialmente svnadmin create) a veces se cuelgan.

Generalmente esto es debido a una carencia de la entropía disponible en el sistema. Subversion pide a APR generar números aleatorios para crear UUIds de tiempo en tiempo, y ciertos sistemas operativos se bloquearán para la aleatoriedad de alta calidad. Usted probablemente necesite configurar el sistema para recolectar entropía de fuentes tales como interrupciones de disco duro y de la red. Consulte las páginas de manual de su sistema, específicamente **random(4)** y **rndcontrol(8)** en cómo efectuar este cambio. Otro trabajo es compilar APR contra `/dev/urandom` en vez de `/dev/random`.

Puedo ver mi repositorio en un navegador web, pero 'svn checkout' me da un error sobre "301 Movido Permanentemente".

Esto significa que su httpd.conf está sin configurar. Generalmente este error ocurre cuando ha definido la "ubicación" virtual para existir dentro de dos alcances al mismo tiempo.

Por ejemplo, si ha exportado un repositorio como `<Location /www/foo>`, pero también ha definido su `DocumentRoot` para ser `/www`, entonces usted está en apuros. Cuando la petición viene para `/www/foo/bar`, apache no sabe si encontrar un fichero *real* llamado `/foo/bar` dentro de su `DocumentRoot`, o si pedir a mod_dav_svn para obtener un fichero `/bar` del repositorio `/www/foo`. Generalmente el caso formado gana, y aparece el error "Movido Permanentemente".

La solución es asegurarse que su repositorio `<Location>` no solape o viva dentro de cualquier área ya exportada como parte normal de intercambio web.

Estoy intentando mirar una versión antigua de mi fichero, pero svn dice algo sobre "path no encontrado".

Una buena característica de Subversion es que el repositorio entiende las copias y los renombrados, y preserva las conexiones históricas. Por ejemplo, si usted copia `/trunk` a `/branches/mybranch`, entones el repositorio entiende que cada fichero en la rama tiene un "predecesor" en el tronco. Ejecutando `svn log --verbose` le mostrará la copia histórica, así que usted puede ver el renombrado:

```
r7932 | jose | 2003-12-03 17:54:02 -0600 (Wed, 03 Dec 2003) | 1 line
Changed paths:
   A /branches/mybranch (from /trunk:7931)
```

Desafortunadamente, mientras el repositorio está enterado de las copias y renombres, casi todos los subcomandos del cliente svn con versión 1.0 *no* están enterados. Comandos como **svn diff**, **svn merge**, y **svn cat** es necesario que entiendan y sigan los renombrados, pero todavía no hace esto. Esto está programado como una característica post-1.0. Por ejemplo, si hace a **svn diff** comparar dos versiones anteriores de `/branches/mybranch/foo.c`, el comando no entenderá automáticamente que la tarea requiere realmente comparar dos versiones de `/trunk/foo.c`, debido al renombrado. En su lugar, usted verá un error sobre cómo la ruta de la rama no existe en las versiones anteriores.

La solución para todos los problemas de este tipo es hacer el trabajo sucio usted mismo. Esto es, *usted* necesita ser conocedorde cualquier ruta renombrada, descubriéndolos usted mismo usando **svn log -v**, y después proporcionelos explícitamente al cliente de svn. Por ejemplo, en vez de ejecutar

```
$ svn diff -r 1000:2000 http://host/repos/branches/mybranch/foo.c
svn: Filesystem has no item
svn: '/branches/mybranch/foo.c' not found in the repository at revision 1000
```

...en su lugar usted debería ejecutar

```
$ svn diff -r1000:2000 http://host/repos/trunk/foo.c
...
```

Apéndice C. WebDAV y autoversionado

WebDAV es una extensión a HTTP, y se está haciendo más y más popular como estándar para compartir archivos. Los sistemas operativos actuales se están haciendo cada vez más compatibles con la web, y muchos de ellos tienen soporte interno para montar directorios compartidos exportados por servidores WebDAV.

Si usted usa Apache/mod_dav_svn como su servidor de red para Subversion, entonces, hasta cierto punto, usted también está usando un servidor WebDAV. Este apéndice brinda nociones básicas sobre la naturaleza de este protocolo, cómo lo usa Subversion, y qué tan bien interopera Subversion con otros programas compatibles con WebDAV.

Conceptos básicos de WebDAV

Esta sección da una introducción muy corta y muy general a las ideas detrás de WebDAV. Debería sentar las bases para entender los problemas de compatibilidad entre los clientes y servidores.

WebDAV sencillo

El RFC 2518 define un conjunto de conceptos y métodos de extensión acompañantes a HTTP 1.1 que hacen de la web un medio de lectura/escritura más universal. La idea básica es que un servidor web compatible con WebDAV puede actuar como un servidor genérico de archivos; los clientes pueden montar directorios compartidos que se comportan de manera muy similar a los directorios compartidos de NFS o SMB.

Sin embargo, es importante notar que el RFC 2518 *no* provee ningún tipo de modelo para control de versiones, a pesar de la “V” en DAV. Los clientes y servidores de WebDAV básico asumen que sólo existe una versión de cada archivo o directorio, y puede ser sobreescrita repetidamente. [1]

Aquí están los conceptos y métodos nuevos introducidos en el WebDAV básico.

Nuevos métodos de escritura

> Más allá del método `PUT` del HTTP estándar (que crea o sobreescribe un recurso web), WebDAV define los nuevos métodos `COPY` y `MOVE` para duplicar o reacomodar recursos.

Colecciones

> Este es simplemente el término usado en WebDAV para el agrupamiento de recursos (URI's). En la mayoría de los casos, es el análogo a un “directorio”. Se puede decir que algo es una colección si termina con un “/”. De la misma manera que los archivos pueden ser creadoscon el método `PUT`, las colecciones son creadas con el nuevo método `MKCOL`.

Propiedades

> Es la misma idea que está presente en Subversion—metadatos asociados a archivos y colecciones. Un cliente puede mostrar u obtener las propiedades asociadas a un recurso con el nuevo método `PROPFIND`, y puede cambiarlas con el método `PROPPATCH`. Algunas propiedades son completamente creadas y controladas por los usuarios (por ejemplo, una propiedad llamada “color”), y otras creadas y administradas por el servidor WebDAV (por ejemplo, la propiedad que contiene la última fecha de modificación de un archivo). Las primeras son llamadas propiedades “muertas” y las segundas propiedades “vivas”.

Bloqueo

> Un servidor WebDAV puede decidir ofrecer una característica de bloqueo a los clientes— esta parte de la especificación es opcional, aunque muchos servidores WebDAV ofrecen esta característica. Si está presente los clientes pueden usar los nuevos métodos `LOCK` y `UNLOCK` para mediar el acceso a un recurso. En la mayoría de los casos estos métodos son usados para crear bloqueos de escritura exclusivos (como se discutió en “La solución bloqueo-modificación-desbloqueo”), aunque también es

[1] Por esta razón, algunas personas se refieren en broma a los clientes genéricos de WebDAV como clientes “WebDA”

posible tener bloqueos de escritura compartidos.

Extensiones DeltaV

Dado que el RFC 2518 dejó por fuera concetos de versionado, se dejó a otro grupo la responsabilidad de escribir el RFC 3253, que añade el versionado a WebDAV. Los clientes y servidores de WebDAV/DeltaV a menudo son llamados sólo clientes y servidores "DeltaV", ya que DeltaV implica la existencia de WebDAV básico.

DeltaV introduce una gran cantidad de acrónimos, pero no deje que eso lo intimide. Las ideas son bastante directas. Aquí están los nuevos conceptos y métodos presentados en DeltaV.

Versionado por recurso

Como CVS y otros sistemas de control de versiones, DeltaV asume que cada recurso tiene un número de estados potencialmente infinito. Un cliente empieza poniendo un recurso bajo control de versiones usando el nuevo método `VERSION-CONTROL`. Éste crea un nuevo recurso de versión controlada (VCR, por sus siglas en inglés). Cada vez que usted cambia el VCR (vía `PUT`,`PROPPATCH`, etc.), un nuevo estado del recurso es creado, llamado recurso de versión (VR por sus siglas en inglés). Los VRs y VCRs son recursos web ordinarios, definidos por URLs. Los VRs específicos también pueden tener nombres amables con el usuario.

Modelo copia de trabajo del lado del servidor

Algunos servidores DeltaV tienen la habilidad de crear un "espacio de trabajo" virtual en el servidor, donde se ejecuta todo el trabajo. Los clientes usan el método `MKWORKSPACE` para crear un área privada, luego indican que quieren cambiar VCRs específicos editándolos, y "registrando su entrada" de nuevo. En términos de HTTP, la secuencia de métodos sería `CHECKOUT`, `PUT`, `CHECKIN`. Después de cada `CHECKIN`, se crea un nuevo VR,y los contenidos de los VCR's editados ahora "apuntan" al último VR. Cada VCR tiene también un recurso "historia", que hace el seguimiento y ordenamiento de varios estados VR.

Modelo copia de trabajo del lado del cliente

Algunos servidores DeltaV también soportan la idea de que el cliente pueda tener una copia privada de trabajo llena de VRs específicos. (Así es como CVS y Subversion trabajan.) Cuando el cliente quiere enviar cambios al servidor, empieza creando una transacción tamporal al servidor (llamada una actividad)con el método `MKACTIVITY`. El cliente ejecuta entonces un `CHECKOUT` sobre caad VR que desea cambiar, lo que crea un número de "recursos de trabajo" temporales en la actividad, que pueden ser modificados usando los métodos `PUT` y `PROPPATCH`. Finalmente, el cliente ejecuta un `CHECKIN` en cad recurso de trabajo, lo que crea un nuevo VR dentro de cada VCR, y la actividad completa es borrada.

Configuraciones

DeltaV le permite definir colecciones flexibles de VCRs llamadas "configuraciones", que no necesariamente corresponden a directorios particulares. El contenido de cada VCR puede hacerse apuntar a un VR específico usando el método `UPDATE`. Una vez la configuración es perfecta, el cliente puede crear un "snapshot" de toda la configuración, llamado "baseline". Los clientes usan los métodos `CHECKOUT` y `CHECKIN` para capturar estados específicos de las configuraciones, de manera muy similar a cuando usan estos métodos para crear estados VR específicos de VCRs.

Extensibilidad

DeltaV define un nuevo método, `REPORT`, que permite al clienet y al servidor llevar a cabo intercambios personalizados de datos. El cliente envía una solicitud `REPORT` con un cuerpo XML adecuadamente etiquetado y lleno de datos personalizados; asumiendo que el servidor entiende el tipo específico del reporte, responde con un cuerpo XML igualmente personalizado. Esta técnica es muy similar a XML-RPC.

Autoversionado

Para muchos, esta es la aplicación "estrella" de DeltaV. Si el servidor DeltaV soporta esta característica, entonces los clientes WebDAV básico (por ejemplo, aquellos que no son compatibles con versionado) aún pueden escribir en el servidor, y el servidor silenciosamente hará el versionado. En el ejemplo más simple, un `PUT` ignorante de parte de un cliente WebDAV básico

puede ser traducido por el servidor como un `CHECKOUT`, `PUT`, `CHECKIN`.

Subversion y DeltaV

Así que ¿qué tan “compatible” es Subversion con otro software DeltaV? En dos palabras: no mucho. Al menos no aún, no en Subversion 1.0.

Mientras que libsvn_ra_dav envía solicitudes DeltaV al servidor, el cliente de Subversion *no* es un cliente DeltaV de propósito general. De hecho, espera ciertas características particulares (especialmente a través de solicitudes `REPORT` personalizadas). Además, mod_dav_svn *no* es un servidor DeltaV de propósito general. Sólo implementa un subconjunto estricto de la especificación DeltaV. Un cliente WebDAV o DeltaV más general puede interoperar bastante bien con él, pero sólo si el cliente opera dentro de los estrechos confines de aquéllas caractrísticas que el servidor ha implementado. El equipo de desarrollo de Subversion planea completar la interoperabilidad general con WebDAV en un lanzamiento futuro de Subversion.

Mapping Subversion to DeltaV

Aquí se presenta una descripción muy general de cómo varias operaciones del cliente de Subversion usan DeltaV. En muchos casos, estas explicaciones son simplificaciones groseras. *No* deberían ser tomadas como un sustituto frente a leer el código fuente de Subversion o hablar con sus desarrolladores.

svn checkout/list

Ejecuta un `PROPFIND` de profundidad 1 en la colección para obtener una lista de los hijos inmediatos. Ejecuta un `GET`(y posiblemente un `PROPFIND`) en cada hijo. Avanza recursivamente dentro de las colecciones y repite.

svn commit

Crea una actividad con `NKACTIVITY`, y hace un `CHECKOUT` de cada ítem que ha cambiado, seguido de un `PUT` de datos nuevos. Finalmente, una solicitud de `MERGE`provoca un `CHECKIN` implícito de todos los recursos de trabajo.

svn update/switch/status/merge/diff

Envía una petición personalizada `REPORT` que describe Send a custom `REPORT` request that describes the mixed-revision (and mixed-url) state of the working copy. The server sends a custom response that describes which items need updating. The client loops over the response, performing `GET` and `PROPFIND` requests as needed. For updates and switches, install the new data in the working copy. For diff and merge commands, compare the data to the working copy, possibly applying changes as local modifications.

Soporte de autoversionado

En el momento de escribir esto, la verdad es que hay muy pocos clientes DeltaV en el mundo; el RFC 3253 aún es relativamente nuevo. Sin embargo, los usuario tienen acceso a clientes “genéricos”, porque casi cada sistema operativo moderno tiene integrado un cliente básico WebDAV. Con esto en mente, los desarrolladores de Subversion se dieron cuenta de que si Subversion 1.0 iba a tener *cualquier* característica de interoperabilidad, el soporte para autoversionado DeltaV sería la mejor aproximación.

Para activar el auotversionado en mod_dav_svn, use la directiva `SVNAutoversioning` dentro del bloque `Location` en el archivo `httpd.conf`, así:

```
<Location /repos>
DAV svn
SVNPath /absolute/path/to/repository
SVNAutoversioning on
</Location>
```

Normalmente, si un cliente WebDAV genérico intentó un `PUT` a una ruta dentro de la ubicación de su repositorio, mod_dav_svn rechazaría la petición. (Normalmente sólo permite este tipo de operaciones en “recursos de trabajo” dentro de “actividades” DeltaV.) Con la opción `SVNAutoversioning` activada, sin embargo, el servidor interpreta la petición `PUT` como un `MKACTIVITY`, `CHECKOUT`, `PUT` y `CHECKIN`. Un mensaje de registro genérico se genera automáticamente, y se crea además una nueva revisión del sistema de archivos

Dado que muchos sistemas operativos ya tienen integradas habilidades WebDAV, el caso de uso para esta característica raya en lo fantástico: imagine una oficina de usuarios ordinarios ejecutando Windows o Mac OS. Cada computador “monta” el repositorio de Subversion, que aparece como una unidad compartida de red cualquiera. Usan el servidor como siempre lo hacen: abren archivos del servidor, los editan, y los guardan de vuelta en el servidor. Pero en esta fantasía, el servidor está versionando todo automáticamente. Después, un administrador del sistema puede usar un cliente de Subversion para buscar y recuperar todas las versiones antigüas.

¿Es esta fantasía real? No mucho. El problema principal es que Subversion 1.0 no tiene ningún tipo de soporte para los métodos `LOCK` o `UNLOCK`. Muchos clientes DAV de sistemas operativos intentan hacer un `LOCK` sobre un recurso abierto directamente de una unidad compartida montada mediante DAV. Por ahora, los usuarios pueden tener que copiar un archivo de la unidad DAV al disco local, etidar el archivo, y copiarlo de vuelta. No es el autoversionado ideal, pero aún hacible.

La Alternativa mod_dav_lock

El módulo mod_dav de Apache es una bestia compleja: entiende y analiza sintácticamente todos los métodos WebDAV y DeltaV, pero depende de un proveedor externo para acceder a los recursos en sí.

En su encarnación más simple, un usuario puede usar mod_dav_sf como un proveedor para mod_dav. mod_dav_fs usa el sistema de archivos ordinario para guardar archivos y directorios, y sólo entiende métodos WebDAV puros, no DeltaV.

Subversion, por otra parte, usa mod_dav_svn como un proveedor para mod_dav. mod_dav_svn entiende todos los métodos WebDAV excepto `LOCK`, y entiende un subconjunto medible de métodos DeltaV. Él accesa los datos en el repositorio Subversion, en vez de hacerlo en el sistema de archivos real. Subversion 1.0 no soporta bloqueo, porque sería realmente difícil de implementar, dado que Subversion usa el modelo copiar-modificar-mezclar. [2]

En Apache httpd-2.0, mod_dav soporta el método `LOCK` llevando la cuenta de los bloqueos en una base de datos privada, asumiendo que el proveedor quiera aceptar esto. En Apache httpd-2.1 o posterior, sin embargo, el soporte de bloqueo se ha puesto en un módulo independiente, mod_dav_lock. Esto le permite a cualquier proveedor de mod_dav hacer uso de la base de datos de bloqueos, incluyendo a mod_dav_svn, aún cuando mod_dav_svn no sabe nada de bloqueo actualmente.

¿Confundido aún?

Resumiendo, mod_dav_lock puede usarse en Apache httpd-2.1 (o posterior) para crear la *ilusión* de que mod_dav_svn está cumpliendo las peticiones `LOCK`. Asegúrese de que mod_dav_lock esté compilado en httpd, o de que está siendo cargado en su `httpd.conf`. Luego simplemente añada la directiva `DAVGenericLockDB` a su archivo de manera similar a ésta:

```
<Location /repos>
DAV svn
SVNPath /absolute/path/to/repository
SVNAutoversioning on
DavGenericLockDB /path/to/store/locks
</Location>
```

Esta técnica es un negocio peligroso; de cierta manera, mod_dav_svn le está mintiendo ahora al cliente WebDAV. El módulo dice aceptar las solicitudes `LOCK`, pero en realidad el bloqueo no está siendo forzado en todos los niveles. Si un segundo cliente WebDAV intenta hacer un `LOCK` sobre el mismo recurso, entonces mod_dav_lock se dará cuenta de ello y denegará (correctamente) la solicitud. Pero no hay absolutamente nada que evite que un cliente Subversion ordinario cambie el archivo vía el comando **svn**

[2] Tal vez algún día Subversion desarrolle un modelo de checkout reservado con bloqueo que pueda vivir en paz con copiar-modificar-mezclar, pero probablemente esto no pase pronto.

commit!. Si usted usa esta técnica, le está dando a los usuarios la oprtunidad de pisotear los cambios de otros. En paticular, un cliente WebDAV podría sobreescribir accidentalmente un cambio enviado por cliente svn normal.

Por otra parte, si usted prepara su entorno cuidadosamente, puede mitigar el riesgo. Por ejemplo, si *todos* sus usuarios están trabajando a través de clientes WebDAV básicos (en vez de clientes svn), entonces todo debería estar bien.

Interoperabilidad de autoversionado

En esta sección describiremos los clientes WebDAV genéricos más comunes (al momento de escribir esto), y qué tan bien operan con un servidor mod_dav_svn usando la directiva `SVNAutoversioning`. El RFC 2518 es un poco largo, y tal vez demasiado flexible. Cada cliente WebDAV se comporta ligeramente diferente, y esto crea problemas ligeramente diferentes.

WebFolders Win32

Windows 98, 2000, y XP tienen un cliente integrado WebDAV conocido como “WebFolders”. En Windows 98, esta característica puede necesitar ser instalada de manera explícita; si está presente, un directorio “WebFolders” aparece directamente dentro de Mi PC. En Windows 2000 y XP, simplemente abra Mis Sitios de Red, y ejecute el icono Añadir Sitio de Red. Cuando se le solicite, ingrese la URL WebDAV. La carpeta compartida aparecerá dentro de Mis Sitios de Red.

Muchas de las operaciones de escritura funcionan bien con un servidor de autoversionado mod_dav_svn, pero hay unos cuantos problemas:

- Si un computador Windows XP es miembro de un dominio NT, parece ser incapaz de conectarse a la carpeta compartida WebDAV. Pide repetidamente el nombre y contraseña, aún cuando el servidor Apache no está presentando un reto de autenticación! Si la máquina no es parte de un dominio NT, entonces la carpeta compartida es montada sin ningún problema.

  Este problema parece surgir de los cambios en la manera en que Windows XP crea accesos directos WebFolder (archivos `.lnk`). Algunas veces reemplaza la URL de la carpeta compartida WebDAV con una ruta Windows “UNC” (Universal Naming Convention). Esto hace que Explorer intente hacer la conexión usando SMB en vez de HTTP.

  Una manera de resolver el problema es crear el acceso directo `.lnk` en un computador Windows 2000 y luego copiar el acceso directo al computador Windows XP. Porbablemente también sería posible “arreglar” el acceso directo usando un editor HEX, si se fuera a hacer ingeniería inversa sobre el formato de archivo `.lnk`.

- Un archivo no puede ser abierto para edición directamente sobre la carpeta compartida; siempre es de sólo lectura. La técnica mod_dav_lock no ayuda, porque los WebFolders no usan el método `LOCK` en absoluto. Sin embargo, el método `copiar, editar, re-copiar`, mencionado anteriormente, funciona. El archivo en la carpeta compartida puede ser sobreescrito exitosamente por una copia editada localmente.

Mac OS X

El sistema operativo de Apple, OS X, tiene un cliente WebDAV integrado. Desde Finder, seleccione el ítem “Conectar a servidor” desde el menú Ir. Ingrese una URL WebDAV, y ésta aparecerá como un disco en el escritorio, como cualquier servidor de archivos. [3]

Desafortunadamente, este cliente se rehúsa a trabajar con autoversionado mod_dav_svn debido a la falta de soporte de `LOCK`. Mac OS X descubre la falta de `LOCK` durante el intercambio de las características HTTP `OPTIONS`, y debido a esto monta el repositorio Subversion como una carpeta compartida de sólo lectura. Desupés de esto, no es posible hacer operaciones de escritura en absoluto. Para montar el repositorio como una carpeta compartida de lectura-escritura, usted *debe* usar el truco con mod_dav_lock como se discutión previamente. Una vez el bloqueo parezca trabajar, la carpeta compartida se comporta bastante bien: los archivos pueden abrirse directamente para lectura/escritura, aunque cada operación de guardado hará que el cliente haga un `PUT` a una ubica-

[3] Los usuarios de Unix también pueden ejecutar **mount -t webdav URL /mountpoint**.

ción temporal, un `DELETE` del archivo original, y un `MOVE` del recurso temporal al nombre de archivo original. ¡Ésas son tres revisiones Subversion nuevas en cada guardado!

Una palabra de advertencia: El cliente WebDAV de OS X puede ser demasiado sensitivo a las redirecciones HTTP. Si usted no puede montar el repositorio en absoluto, puede que necesite habilitar la directiva `BrowserMatch` en su `httpd.conf`:

```
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
```

Unix: Nautilus 2

Nautilus es el administrador/explorador de archivos oficial del escritorio GNOME. Su página principal está en `http://www.gnome.org/projects/nautilus/`. Sólo con escribir una URL WebDAV en la ventana de Nautilus, la carpeta DAV aparece como un sistema de archivos local.

En general, Nautilus 2 trabaja razonablemente bien con un mod_dav_svn que haga autoversionado, con las siguientes precauciones:

- Cualquier archivo abierto directamente desde la carpeta compartida es tratado como de sólo lectura. Aún el truco con mod_dav_lock parece no tener efecto. Parece que Nautilus nunca llama el método `LOCK` en absoluto. El truco “copiar localmente, editar, copiar de vuelta” funciona, sin embargo. Desafortunadamente, Nautilus sobreescribe el archivo viejo llamando a `DELETE` primero, lo que crea una revisión extra.
- Cuando se sobreescribe o se crea un archivo, Nautilus hace primero un `PUT` de un archivo vacío, y luego lo sobreescribe con un segundo `PUT`. Esto crea dos revisiones en el sistema de archivos Subversion, en vez de una sola.
- Cuando se borra una colección, se llama un `DELETE` HTTP en cada hijo individual en vez de hacerlo sobre toda la colección. Esto crea un montón de revisiones nuevas.

Linux davfs2

Linux davfs2 es un módulo de sistema de archivos para el kernel de Linux, cuyo desarrollo se ubica en `http://dav.sourceforge.net/`. Una vez instalado, una carpeta WebDAV compartida puede ser montada con el comando estándar de Linux **mount**.

Se dice en las calles que este cliente DAV no funciona en absoluto con el autoversionado de mod_dav_svn. Cada intento de escribir al servidor es precedido por una solicitud `LOCK`, que mod_dav_svn, no soporta. En este momento, no hay datos que indiquen si el uso de mod_dav_lock resuelva este problema.

Apéndice D. Herramientas de terceras partes

El diseño modular de Subversion (cubierto en "Diseño de librería por capas") y la disponibilidad de atascamientos del lenguaje (según lo descrito en "Usando lenguajes distintos de C y C++") lo hacen un candidato para ser usado como una extensión o añadido a otras piezas de software. En este apéndice, le introduciremos brevemente a algunas de las muchas herramientas de terceras partes que están usando la funcionalidad de Subversion bajo la capucha.

Para una versión más recientemente actualizada de esta información, compruebe la página de Links en el website de Subversion (`http://subversion.tigris.org/project_links.html`).

Clientes y módulos

AnkhSVN (`http://ankhsvn.tigris.org/`)

: Extensión de Subversion para Microsoft Visual Studio .NET

JSVN (`http://jsvn.alternatecomputing.com/`)

: Cliente Java de Subversion, incluyendo un módulo para IDEA

psvn.el (`http://xsteve.nit.at/prg/vc_svn/`)

: Interfaz de Subversion para emacs

RapidSVN (`http://rapidsvn.tigris.org/`)

: GUI multiplataforma de Subversion, basado en las librerías WxPython

Subclipse (`http://subclipse.tigris.org/`)

: Módulo de Subversion para el entorno Eclipse

Subway (`http://nidaros.homedns.org/subway/`)

: Proveedor de Microsoft SCC para Subversion

sourcecross.org (`http://www.sourcecross.org/`)

: Proveedor de Microsoft SCC para Subversion

Supervision (`http://supervision.tigris.org/`)

: Cliente visual Java/Swing para Subversion

Sven (`http://www.nikwest.de/Software/#SvenOverview`)

: GUI nativo para Subversion usando la infraestructura Cocoa de Mac OS X

Svn4Eclipse (`http://svn4eclipse.tigris.org/`)

: Módulo Subversion para el IDE Eclipse

Svn-Up (`http://svnup.tigris.org/`)

GUI basado en Java para Subversion y módulo para el IDE IDEA

TortoiseSVN (`http://tortoisesvn.tigris.org/`)

Cliente Subversion, implementado como una extensión del intérprete de comandos de Microsoft Windows

WorkBench (`http://pysvn.tigris.org/`)

Interfaz de desarrollo de software multiplataforma basado en Python para Subversion

Language Bindings

PySVN (`http://pysvn.tigris.org/`)

Interfaz para Python orientada a objetos con la API del cliente de Subversion.

Subversion (`http://subversion.tigris.org/`)

Interfaces a la API de Subversion para Python, Perl y Java, reflejando la API del núcleo en C

SVNCPP (`http://rapidsvn.tigris.org/`)

Interfaz para C++ orientada a objetos con la API del cliente de Subversion

Conversores de repositorios

cvs2svn (`http://cvs2svn.tigris.org/`)

Conversor CVS a Subversion

vss2svn (`http://vss2svn.tigris.org/`)

Conversor Microsoft SourceSafe a Subversion

Módulo Subversion VCP (`http://svn.clkao.org/revml/branches/svn-perl/`)

Módulo VCP para CVS a Subversion

Herramientas de mayor nivel

Kwiki (`http://www.kwiki.org/`)

Wiki con motor Subversion para copias de seguridad

Subissue (`http://subissue.tigris.org/`)

Lleva el registro de problemas directamente en un repositorio de Subversion

Subwiki (`http://subwiki.tigris.org/`)

Wiki que usa Subversion como su repositorio de datos

svk (`http://svk.elixus.org/`)

Sistema de control de versiones descentralizado basada en Subversion

submaster (`http://www.rocklinux.org/submaster.html`)

Sistema para desarrollo de software distribuido, basado en Subversion

Herramientas de exploración de repositorios

SVN::Web (`http://svn.elixus.org/repos/member/clkao/`)

Interfaz Web basada en Perl para repositorios Subversion

ViewCVS (`http://viewcvs.sourceforge.net/`)

Script CGI basado en Python para hacer búsquedas en repositorios de CVS y Subversion

WebSVN (`http://websvn.tigris.org/`)

Buscador para repositorios Subversion basado en PHP

Trac (`http://projects.edgewall.com/trac`)

Gestor de proyectos de software minimalista basado en web y sistema de seguimiento de errores/problemas con interfaz de control de versiones y con soporte Wiki integrado

Apéndice E. Sobre esta traducción

La traducción que tiene en sus manos es fruto de la colaboración desinteresada de varias personas. Su esfuerzo y dedicación se ven plasmados a lo largo de todo el libro, y es en este apéndice, que no existe en la versión original, donde se reconoce su trabajo. Puede obtener la última versión de este libro, e información sobre el proyecto de traducción en `http://svnbook.red-bean.com/index.es.html`.

Origen del proyecto

El 15 de septiembre del 2004, Grzegorz Adam Hankiewicz y Rubén Gómez anunciaron en la lista de correo de desarrolladores de Subversion el inicio del proyecto de traducción del libro de Subversion al español[1]. Desde aquel mensaje el repositorio ha cambiado un par de veces de ubicación, se creó una lista de correo para discutir temas relacionados con la traducción del libro, etc, etc.

Desde entonces, varias otras personas se han unido al proyecto contribuyendo traducciones, revisiones, críticas y sugerencias. No considere esta traducción como un producto finalizado: aparte de posibles fallos en la traducción, día a día se sigue mejorando la versión inglesa que puede obtener via web desde `http://svnbook.red-bean.com/`, y obviamente hay que adaptar también la traducción. Así que le animamos a que se pase por la lista de correo (más información debajo) para comentarnos su opinión.

Quienes somos

A continuación mostramos una lista ordenada alfabéticamente por nombre de las personas que han colaborado con el proyecto, indicando entre paréntesis la naturaleza de su aportación:

- Ariel Arjona (traducción)
- Diego Brouard (traducción)
- Domingo Suárez Torres (traducción)
- Federico Edelman (traducción)
- Grzegorz Adam Hankiewicz (coordinación, traducción)
- Javier Rojas (traducción)
- Rubén Gómez (traducción)

Listas de correo

La mejor forma de ponerse en contacto con los participantes del proyecto es mandando un email a la lista de correo creada para

[1] Posiblemente pueda leer todavía este mensaje en el archivo público `http://subversion.tigris.org/servlets/ReadMsg?list=dev&msgNo=77338`.

coordinar las traducciones relacionadas con Subversion: <l10n-es@subversion.tigris.org>. No necesita suscribirse a la lista para poder mandar mensajes, pero en ese caso serán moderados y puede que tarden uno o dos días en llegar al resto de los suscriptores.

Para apuntarse a esta lista de correo, mande un email a la dirección <l10n-es-subscribe@subversion.tigris.org>. Para desapuntarse, envíe un email a la dirección <l10n-es-unsubscribe@subversion.tigris.org>. El servicio de lista de correo es automático, y en ambos casos recibirá instrucciones para confirmar su acción.

Hay otras listas de correo alojadas en el servidor principal de Subversion. Puede encontrar información sobre éstas e instrucciones en la dirección http://subversion.tigris.org/servlets/ProjectMailingListList.

Apéndice F. Copyright

1. Definitions

   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.

   d. "Original Author" means the individual or entity who created the Work.

   e. "Work" means the copyrightable work of authorship offered under the terms of this License.

   f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   b. to create and reproduce Derivative Works;

   c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

e.

For the avoidance of doubt, where the work is a musical composition:

i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or

Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/.

==========

Colophon

Etc.