Desarrollo de Código Abierto
Con
CVS

Karl Fogel <kfogel@red-bean.com>

Este manual describe como usar y administrar CVS (Concurrent Versions System). Es parte un trabajo mayor titulado *Desarrollo de Código Abierto con CVS*; por favor, lea la introducción para más detalles.

Esta es la versión 1.21 de este manual.

Short Contents

Table of Contents

Introduccion

Esto es un conjunto de capítulos de difusión libre y disponibles online sobre el uso de CVS (Sistema Concurrente de Versiones) para la colaboración y el control de versiones. Cubre desde la instalación de CVS hasta el uso avanzado y la administración. Está dirigido a cualquiera que use o planee usar CVS.

Estos capítulos están extraídos de una trabajo mayor llamado *Desarrollo de Código Abierto con CVS* (publicado por The Coriolis Group (`http://www.coriolis.com/`), ISBN 1-57610-490-7). El resto del libro – capítulos 1, 3, 5 y 7 – trata de los desafíos y temas filosóficos de ejecutar un proyecto de Código Abierto usando CVS.

Aunque los capítulos disponibles libremente en sí mismos constituyen un libro de CVS completo, esperamos que le gusten lo suficiente como para comprar una copia impresa del libro entero! Puede pedirlo directamente del editor en `http://www.coriolis.com/bookstore/bookdetail.cfm?id=1576104907`.

Estos capítulos están disponibles bajo la GNU General Public License (`http://www.gnu.org/copyleft/gpl.html`). Para más información sobre el software libre en general, visite `http://www.gnu.org/`, and particularly `http://www.gnu.org/philosophy/free-sw.html`.

Para mandar comentarios o errores sobre este material, por favor, mande un email a `bug-cvsbook@red-bean.com`. Para noticias y actualizaciones, visite `http://cvsbook.red-bean.com/`.

Una introduccion a CVS

> *No puedo imaginarme programar sin él.. sería como saltar en paracaídas sin un paracaídas!*
>
> *–Brian Fitzpatrick hablando de CVS*

Este capítulo presenta los fundamentos en los que se basa CVS, ofreciendo a continuación un recorrido detallado por el uso cotidiano de CVS. Los conceptos se presentan de forma secuencial, así que si es Ud. un novato en CVS, la mejor manera de leer este capítulo será empezar por el principio e ir leyéndolo paso a paso, sin saltarse nada.

Conceptos Básicos

Si nunca ha usado antes CVS (o ningún otro sistema de control de versiones), es fácil desconcertarse con algunas de las suposiciones que éste hace. Lo que parece causar más confusión inicialmente sobre CVS, es que se usa para dos cosas que aparentemente no guardan relación alguna: guardar registros, y al mismo tiempo hacer posible la colaboración. Sin embargo, se da el caso de que estas dos funciones están estrechamente relacionadas.

Se hizo necesario guardar registros porque las personas querían comparar el estado actual de un programa con el estado en el que encontraba en un momento dado en el pasado. Por ejemplo, en el transcurso normal de la implantación de una nueva función, un desarrollador puede llevar el programa a un estado en el que resulta inutilizable, estado que posiblemente perdurará hasta que la implantación de la nueva función esté casi hecha. Por desgracia, se da la casualidad de que éste suele ser precisamente el momento en el que alguien informa de un fallo en la última versión distribuida al público; para solucionar el problema (que podría estar también presente en la versión actual de las fuentes), el programa ha de volver de nuevo a un estado utilizable.

Restaurar un estado determinado es tarea fácil si el historial del código fuente se mantiene bajo CVS. El desarrollador puede simplemente decir: "Dame el programa tal como estaba hace tres semanas", o quizás, "Dame el programa en el estado en el que se encontraba en el momento de hacer nuestra última distribución pública". Si nunca ha disfrutado de este cómodo acceso a "fotografías" históricas, posiblemente se sorprenda de la rapidez con la que llegará a depender de él. En mi caso, personalmente ahora siempre uso control de revisiones en mis proyectos de programación, puesto que es algo que me ha salvado en multitud de ocasiones.

Para comprender cómo está ésto relacionado con la colaboración, deberemos observar con más detalle el mecanismo ofrecido por CVS para ayudar a muchas personas a trabajar en un mismo proyecto. Pero, antes de que lleguemos ahí, echemos un vistazo al mecanismo que CVS *no* proporciona (o que, por lo menos, no fomenta): el bloqueo de ficheros. Si ha usado algún otro sistema de control de versiones, quizás esté familiarizado con el modelo de desarrollo bloquear-modificar-desbloquear, en el que un desarrollador obtiene primero un acceso exclusivo de escritura (un bloqueo) sobre el fichero que va a editar, hace los cambios, y a continuación elimina el bloqueo para permitir que otros desarrolladores puedan acceder al fichero. Si alguien tiene un bloqueo establecido sobre un fichero, esa persona deberá "liberarlo" para que usted pueda bloquear el fichero y comenzar a hacer sus

cambios (en algunas implantaciones usted podría "robarle" el bloqueo, pero con frecuencia esto viene a ser una desagradable sorpresa para la otra persona, y en ningún caso una práctica aconsejable!).

Este sistema es factible cuando los desarrolladores se conocen, saben quién se propone hacer qué en un momento dado, y pueden comunicarse con los demás rápidamente en caso de que alguien no pueda trabajar por tener limitado su acceso. Sin embargo, si el grupo de desarrolladores se hace demasiado grande o no es posible una comunicación fluida entre ellos, gestionar cuestiones de bloqueo de ficheros comenzará a restar tiempo al desarrollo del código, para convertirse en un problema constante que puede ser contraproducente para el normal discurrir del proyecto.

CVS propone una solución intermedia: en lugar de obligar a los desarrolladores a coordinarse entre sí para evitar conflictos, CVS les permite editar el código de forma simultánea, asume la tarea de integrar todos los cambios, y guarda un registro de todos los conflictos que vayan surgiendo. El proceso utiliza el modelo copiar-modificar-fusionar, que viene a funcionar de la forma siguiente:

1. El desarrollador A solicita a CVS una copia de trabajo, esto es, un árbol de directorios que contiene los ficheros que conforman el proyecto. Esta operación es también conocida como "obtener una copia" (comando "checkout"), y es como tomar un libro prestado de una biblioteca.
2. El desarrollador A edita libremente su copia de trabajo. Al mismo tiempo, otros desarrolladores pueden estar atareados con sus propias copias de trabajo. Puesto que todas son copias separadas, no hay interferencias: es como si todos los desarrolladores tuvieran su propia copia del mismo libro, y todos estuvieran trabajando anotando comentarios en los márgenes o reescribiendo determinadas páginas de forma independiente.
3. El desarrollador A termina sus cambios y los envía (comando "commit") a CVS junto con un informe de cambios, que es un comentario que explica la naturaleza y propósito de los cambios que ha realizado. Esto es el equivalente a informar a la biblioteca de los cambios que ha hecho al libro y el porqué de los mismos. Entonces, la biblioteca incorpora estos cambios a la "copia maestra", donde se guardan de forma permanente.
4. Mientras tanto, y también por medio de CVS, otros desarrolladores pueden consultar a la biblioteca para ver si la copia maestra ha cambiado recientemente, en cuyo caso CVS actualizará automáticamente sus copias de trabajo personales. (Esta parte es mágica y maravillosa, y espero que sepa apreciarla. Imagine lo diferente que sería el mundo si los libros de verdad funcionasen de esta forma!)

Por lo que concierne a CVS, todos los desarrolladores de un proyecto son iguales. La decisión de cuándo actualizar o cuándo enviar al almacén es un tema de preferencias personales o de política establecida entre los miembros que participan en el proyecto. Una de las estrategias más comunes para proyectos de programación es la de siempre actualizar antes de empezar a trabajar en un cambio importante, y enviar los cambios sólo cuando éstos hayan sido finalizados y probados, a fin de que la copia principal se mantenga en todo momento en un estado "ejecutable".

Quizás se esté preguntando lo que ocurre cuando los desarrolladores A y B, cada uno trabajando en su copia de trabajo personal, hacen distintos cambios al mismo trozo de texto y después ambos envían sus cambios. Esto se conoce como *conflicto*, y CVS se percata del

mismo en cuanto el desarrollador B intenta enviar sus cambios: en lugar de permitir que el desarrollador B proceda, CVS anuncia que ha descubierto un conflicto y coloca marcadores de conflicto (marcas de texto fácilmente reconocibles) en el lugar de su copia local donde se ha descubierto el conflicto. En ese lugar se mostrarán ambos paquetes de cambios, convenientemente ordenados para hacer fácil su comparación. El desarrollador B deberá entonces solucionar el problema y enviar una nueva revisión con el conflicto resuelto. Quizás los dos desarrolladores deban hablar entre ellos para solucionar el problema; CVS sólo avisa a los desarrolladores de que hay un conflicto, dejando bajo su responsabilidad la tarea de resolverlo.

Y qué hay de la copia maestra? En terminología oficial de CVS, se la conoce como "repositorio" del proyecto, y es simplemente un árbol de ficheros guardado en un servidor central. Sin entrar en mucho detalle sobre su estructura (pero lea ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩), veamos lo que el repositorio debe hacer para cumplir con los requisitos del ciclo copiar-enviar-actualizar. Considere el siguiente caso:

1. Dos desarrolladores, A y B, obtienen una copia de trabajo de un proyecto al mismo tiempo. El proyecto se encuentra en su punto de partida: nadie ha hecho todavía cambio alguno, así que todos los ficheros están todavía en su estado original e impoluto.
2. El desarrollador A empieza de inmediato a trabajar, y pronto envía su primer paquete de cambios.
3. Mientras tanto, el desarrollador B está viendo la televisión.
4. El desarrollador A, trabajando como si el mundo se acabase al día siguiente, envía su segundo paquete de cambios. En este momento, el historial del repositorio contiene los ficheros originales, seguidos por el primer paquete de cambios introducidos por A, que a su vez han ido seguidos por estos últimos cambios.
5. Mientras tanto, el desarrollador B está jugando a su videojuego favorito.
6. De pronto, el desarrollador C se une al proyecto y obtiene su copia de trabajo del repositorio. La copia del desarrollador C refleja los dos primeros paquetes de cambios de A, puesto que ya estaban en el repositorio cuando C obtuvo su copia.
7. El desarrollador A, que ha estado programando como un poseso, completa y envía su tercer paquete de cambios.
8. Por último, e ignorando la actividad frenética habida recientemente, B decide que es hora de empezar a trabajar. No se molesta en actualizar su copia; simplemente comienza a editar ficheros, algunos de los cuales pueden ser ficheros en los que A ha estado trabajando. Un poco más tarde, el desarrollador B envía sus primeros cambios.

Llegados a este punto, pueden suceder dos cosas. Si ninguno de los ficheros editados por B han sido editados por A, el envío tiene éxito. Sin embargo, si CVS percibe que algunos de los ficheros de B están pasados de fecha con respecto a las últimas copias disponibles en el repositorio, y todos esos ficheros han sido también cambiados por B en su copia de trabajo, CVS informa a B de que debe hacer una actualización antes de enviar estos ficheros.

Cuando el desarrollador B efectúa la actualización, CVS reune todos los cambios realizados por A en la copia local de los ficheros de B. Parte del trabajo de A puede entrar en conflicto con los cambios no enviados por B, mientras que otros pueden no hacerlo. Aquellas partes que no lo hacen son simplemente aplicadas en las copias de B, sin más, pero los cambios que supongan un conflicto deberán ser resueltos por B para poder ser enviados.

Si el desarrollador C efectúa ahora una actualización, recibirá del repositorio algunos cambios nuevos, que serán aquéllos pertenecientes al tercer envío de A, y los pertenecientes al primero *con éxito* de B (que en realidad podrían proceder del segundo intento de B de enviar, asumiendo que el primer intento de B tuviese como resultado el que B se viera obligado a resolver algún conflicto).

Para que CVS pueda servir los cambios en la secuencia correcta a los desarrolladores cuyas copias de trabajo puedan no estar sincronizadas en mayor o menor grado, el repositorio necesita guardar todos los envíos recibidos desde el comienzo del proyecto. En la práctica, el repositorio de CVS los guarda todos en ficheros de diferencias (también llamados "diffs") sucesivos. Así pues, incluso para una copia de trabajo muy antigua, CVS es capaz de establecer las diferencias entre la copia de trabajo y el estado actual del repositorio, y es por tanto capaz de actualizar la copia de trabajo de una forma eficiente. Esto hace que los desarrolladores puedan en cualquier momento revisar fácilmente el historial del proyecto, y conseguir copias de trabajo tan antiguas como sea necesario.

A pesar de que, estrictamente hablando, el repositorio podría conseguir los mismos resultados por otros medios, en la práctica guardar ficheros de diferencias es una forma simple e intuitiva de implantar la funcionalidad necesaria. Además, este método tiene la ventaja añadida de que, usando apropiadamente el programa "patch", CVS puede reconstruir cualquier estado previo del árbol de ficheros y, por tanto, llevar una copia de trabajo de un estado a otro. Esto permite que cualquiera pueda obtener una copia del proyecto tal y como era en un momento determinado, a la vez que permite mostrar las diferencias, en formato diff, entre dos estados del árbol sin afectar a la copia de trabajo de nadie.

Por lo tanto, las mismas funcionalidades que son necesarias para dar un útil acceso al historial del proyecto también resultan útiles para proporcionar a un grupo de desarrolladores descentralizado y descoordinado la posibilidad de colaborar en el proyecto.

Por ahora, puede ignorar todos los detalles de cómo configurar un repositorio, administrar el acceso a los usuarios, y navegar por formatos de fichero específicos de CVS (los cuales se cubrirán en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩); de momento, nos centraremos en cómo hacer cambios en una copia de trabajo.

Pero antes, aquí va una rápida explicación de los términos:

Revisión Un cambio aplicado y registrado en el historial de un fichero o conjunto de ficheros. Una revisión es una "instantánea" de un proyecto que cambia constantemente.

Repositorio La copia maestra en la que CVS guarda el historial de revisiones al completo efectuadas en un proyecto. Cada proyecto tiene exactamente *un* repositorio.

Copia de trabajo La copia en la que puede de hecho hacer cambios al proyecto. Puede haber muchas copias de trabajo de un proyecto dado. Por regla general, cada desarrollador tiene su propia copia de trabajo.

Obtener una copia ("check out") Solicitar una copia de trabajo al repositorio. Su copia de trabajo refleja el estado del proyecto en el momento de obtenerla; cuando Ud. y otros desarrolladores hacen cambios, deben enviarlos ("commit") y actualizarlos ("update") tanto para "publicar" sus cambios como para ver los que han hecho los demás.

Enviar ("commit") Enviar cambios de su copia local al repositorio central. También conocido como *check-in*.

Informe de cambios Un comentario que se adjunta a una revisión cuando ésta se envía, describiendo los cambios realizados. Otros pueden leer los informes de cambios para obtener un resumen de lo que ha estado sucediendo en un proyecto.

Actualizar ("update") Incorporar a su copia de trabajo los cambios que otros han hecho y están presentes en el repositorio, y comprobar si su copia de trabajo tiene algún cambio que no ha enviado todavía. Tenga cuidado y no confunda esto con el envío; son operaciones complementarias. Recuerde, lo que hace una actualización es sincronizar su copia de trabajo con la copia presente en el repositorio.

Conflicto La situación que se da cuando dos desarrolladores intentan enviar cambios que han hecho al mismo pasaje de un fichero. CVS se da cuenta de ello e informa del conflicto, pero son los desarrolladores quienes tienen que resolverlo.

Un día con CVS

Esta sección describe algunas operaciones básicas de CVS, para a continuación presentarle un ejemplo de sesión de trabajo que cubre el uso que suele hacerse de CVS. A medida que vayamos avanzando por esta guía, empezaremos también a ver cómo funciona CVS interiormente.

Si bien no necesita comprender todos y cada uno de los detalles de CVS para poder utilizarlo, unos conocimientos básicos de cómo funciona le serán de inestimable ayuda a la hora de elegir la mejor forma de conseguir un resultado. CVS se parece más a una bicicleta que a un coche, en el sentido de que sus mecanismos son completamente transparentes para quien le interese examinarlos. Al igual que con una bicicleta, puede subirse encima y empezar a pedalear inmediatamente; sin embargo, si se toma algún tiempo para aprender cómo funciona el cambio de marchas, será capaz de utilizarlo de forma mucho más eficaz. (En el caso de CVS, no estoy seguro de si la transparencia fue un criterio de diseño deliberado o algo accidental, pero parece que es una propiedad compartida con muchos otros programas libres. Las implantaciones que son visibles desde fuera tienen la ventaja de fomentar el que los usuarios contribuyan [a mejorar el software], exponiéndoles desde el primer momento el funcionamiento interno del sistema.)

Convenciones empleadas en este Recorrido

Este recorrido tiene lugar en un entorno UNIX. CVS también funciona en sistemas operativos Windows y Macintosh, y Tim Endres de Ice Engineering ha escrito incluso un cliente en Java (véase http://www.trustice.com/java/jcvs), que puede ejecutarse en cualquier lugar donde corra Java. Sin embargo, voy a suponer que la mayoría de los usuarios de CVS -tanto actuales como potenciales- están seguramente trabajando en un entorno UNIX basado en línea de comandos. Si usted no figura entre éstos, los ejemplos en el recorrido deberían ser fáciles de adaptar a otros interfaces. Una vez que entienda los conceptos, podrá sentarse delante de cualquier interfaz de CVS y empezar a trabajar con él (créame, yo lo he hecho muchas veces).

Los ejemplos que se presentan en este recorrido están orientados a personas que van a usar CVS para trabajar en proyectos de programación. Sin embargo, el uso de CVS es aplicable a todo tipo de documentos de texto, no sólo a código fuente.

Esta guía también asume que tiene CVS ya instalado (por omisión, está ya presente en muchos de los sistemas libres UNIX más populares, así que puede que lo tenga ya instalado sin saberlo) y que ya dispone de acceso a un repositorio. Incluso si no es así, puede beneficiarse de la lectura de esta guía. En ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩, aprenderá cómo instalar CVS y configurar repositorios.

Suponiendo que CVS esté ya instalado, debería tomarse un momento para encontrar el manual de CVS en línea. Se conoce familiarmente como el "Cederqvist" (tomando el apellido de Per Cederqvist, su autor original), viene incluido con la fuente de CVS y viene a ser por lo general la referencia más actualizada que se encuentra disponible. Está escrito en formato Texinfo y debería estar disponible en sistemas Unix en la jerarquía de documentación "Info". Puede leerlo con el programa de de comandos "info":

```
floss$ info cvs
```

o bien pulsando Ctrl+H y después escribiendo "i" (o Esc+x info) dentro de Emacs. Si ninguno de estos métodos funciona para usted, consulte a su experto local en Unix (o mire ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ para los problemas relacionados con la instalación). Seguramente querrá tener el Cederqvist a mano si va a usar CVS regularmente.

Invocación de CVS

CVS es un sólo programa, pero puede hacer muchas cosas diferentes: actualizar, enviar, ramificar, diferenciar, etc.. Cuando invoque a CVS deberá especificar qué operación desea realizar. Así pues, el formato de invocación de CVS viene a ser:

```
floss$ cvs comando
```

Por ejemplo, puede usar

```
floss$ cvs update
floss$ cvs diff
floss$ cvs commit
```

etcétera. (No se moleste de momento en intentar ejecutar ninguna de estas órdenes, puesto que no harán nada mientras no disponga de una copia de trabajo; pronto llegaremos a ese punto.)

Tanto CVS como sus comandos pueden admitir opciones. Las opciones que afectan al comportamiento de CVS, independientemente del comando que se ejecute, se llaman opciones globales, mientras que las opciones que son específicas de los comandos se llaman simplemente opciones de comando. Las opciones globales siempre van a la izquierda del comando, mientras que las opciones de comando van a la derecha. Así, en

```
floss$ cvs -Q update -p
```

-Q es una opción global, y -p es una opción del comando. (Si siente curiosidad, -Q significa "en silencio", es decir, esta opción eliminaría toda salida de diagnóstico y mostraría los mensajes de error únicamente si el comando no puede cumplir con su cometido por alguna razón; por su parte, -p forzaría el envío de los resultados de la actualización a la salida estándar, en lugar de hacerlo a los ficheros).

Acceder a un Repositorio

Antes de hacer nada, deberá decirle a CVS dónde se encuentra el repositorio al que desea acceder. Esto no es problema si ya ha obtenido una copia de trabajo de ese repositorio, dado que cualquier copia de trabajo sabe de qué repositorio procede, con lo cual CVS puede deducir automáticamente el repositorio al que corresponde una copia de trabajo cualquiera. Sin embargo, vamos a suponer que no dispone usted todavía de una copia de trabajo, así que tiene que decirle a CVS explícitamente dónde ir. Esto se hace con la opción global -d (de "directorio", una abreviatura para la que hay una justificación histórica, aunque la -r de "repositorio" hubiera sido mejor), seguido por la senda que apunta al repositorio.

Por ejemplo, asumiendo que el repositorio se encuentra en el sistema local en /usr/local/cvs (un lugar bastante estándar),

```
floss$ cvs -d /usr/local/cvs comando
```

En muchos casos, sin embargo, el repositorio estará en otra máquina, y por tanto deberá usar la red para llegar hasta él. CVS ofrece varios métodos de acceso; cuál de ellos será el que utilice es algo que depende fundamentalmente de los requisitos de seguridad de la máquina en la que se encuentra alojado el repositorio (a la que, de ahora en adelante, nos referiremos como "el servidor"). La configuración del servidor para permitir varios métodos de acceso remoto se comenta en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩; aquí trataremos sólo la parte de cliente.

Afortunadamente, todos los métodos de acceso remoto comparten una misma sintaxis de invocación. En general, para especificar un repositorio remoto y no uno local, lo que hará es utilizar una senda más larga. Primero debe indicar al método de acceso, delimitado en cada lado por símbolos de dos puntos, seguido del nombre de usuario y el nombre del servidor unidos por el símbolo `@`, otros dos puntos de separación, y, finalmente, la senda del directorio del repositorio en el servidor.

Veamos el método de acceso *pserver*, que significa "servidor autentificado por clave":

```
floss$ cvs -d :pserver:jluis@cvs.foobar.com:/usr/local/cvs login
(Logging in to jluis@cvs.foobar.com)
CVS password: (introduzca aquí su contraseña de CVS)
floss$
```

La larga senda del repositorio que sigue a la opción -d ha ordenado a CVS que use el método de acceso pserver, con el nombre de usuario jluis, en el servidor cvs.foobar.com, que tiene un repositorio CVS en /usr/local/cvs. Por cierto, no hay ninguna razón para que el nombre del servidor sea "cvs.algun_lugar.com"; esto es simplemente una convención común, pero podría haber sido también:

```
floss$ cvs -d :pserver:jluis@fish.foobar.org:/usr/local/cvs comando
```

El comando que se ejecutó en nuestro ejemplo fue "login", que verifica que dispone usted de autorización para trabajar en este repositorio. CVS le pedirá una contraseña, contactando a continuación con el servidor para verificarla. Siguiendo la costumbre Unix, "cvs login" no devolverá ninguna información adicional si la operación tiene éxito, aunque sí mostrará un mensaje de error si algo sale mal (por ejemplo, si la contraseña es incorrecta).

Sólo tiene que autentificarse una vez desde su máquina local ante un servidor CVS. Una vez que el proceso de autentificación tiene éxito, CVS guarda la contraseña en su directorio personal, en un fichero llamado .cvspass. CVS consultará este fichero cada vez que se conecte

al repositorio a través del método pserver, así que sólo tiene que ejecutar "login" la primera vez que acceda a un determinado servidor de CVS desde un sistema cliente particular. Por supuesto, puede volver a ejecutar cvs login en cualquier momento, si por ejemplo se hubiera cambiado la contraseña.

Observación: pserver es en este momento el único método de acceso que requiere un proceso de autentificación inicial como éste; con el resto de métodos de acceso puede empezar a ejecutar comandos de CVS inmediatamente.

Una vez que ha guardado la información de autentificación en su fichero .cvspass, puede ejecutar otros comandos de CVS utilizando la misma sintaxis en la línea de comando:

```
floss$ cvs -d :pserver:jluis@cvs.foobar.com:/usr/local/cvs comando
```

Hacer que pserver funcione en Windows puede requerir un paso adicional. Windows carece del concepto Unix de un directorio personal, así que CVS no sabe dónde poner el fichero .cvspass; deberá especificar para ello un lugar concreto. Generalmente se indica la raíz de la unidad C: como el directorio personal:

```
C:\WINDOWS> set HOME=C:
C:\WINDOWS> cvs -d :pserver:jluis@cvs.foobar.com:/usr/local/cvs login
(Logging in to jluis@cvs.foobar.com)
CVS password: (introduzca aquí su contraseña)
C:\WINDOWS>
```

Cualquier carpeta existente en la jerarquía de ficheros resultará válida, aunque posiblemente prefiera evitar utilizar unidades de red, dado que el contenido de su fichero .cvspass sería entonces visible para cualquiera que pudiese acceder a esa unidad.

Además de pserver, CVS soporta también los métodos ext -que utiliza un programa de conexión externo. como rsh ó ssh-, kserver -para el sistema de seguridad Kerberos versión 4-, y gserver, que usa el GSSAPI, esto es, el API de Generic Security Services, y también las versiones 5 y posteriores de Kerberos. Todos estos métodos son similares a pserver, si bien cada uno presenta sus propias idiosincrasias.

De ellos, el método `ext` es probablemente el que más se usa habitualmente. Si tiene la posibilidad de conectarse a un servidor mediante rsh o ssh, puede usar el método `ext`. Puede probarlo de esta forma:

```
floss$ rsh -l jluis cvs.foobar.com
Password:  (introduzca aquí su contraseña de usuario)
```

Bien, vamos a asumir que ha entrado y salido con éxito del servidor con rsh, así que ahora está de nuevo en el sistema cliente original:

```
floss$ CVS_RSH=rsh; export CVS_RSH
floss$ cvs -d :ext:jluis@cvs.foobar.com:/usr/local/cvs comando
```

La primera línea (empleando la sintaxis del shell Bourne de Unix) da a la variable de entorno CVS_RSH el valor rsh, que le dice a CVS que utilice el programa rsh para conectarse. La segunda línea puede ser cualquier comando de CVS; se le solicitará su contraseña para que CVS pueda conectarse con el servidor.

Si está en el shell C en lugar del shell Bourne, pruebe esto:

```
floss% setenv CVS_RSH rsh
```

y para Windows, pruebe esto:

```
C:\WINDOWS> set CVS_RSH=rsh
```

El resto de esta guía empleará la sintaxis Bourne; adapte los ejemplos a su entorno como necesite.

Para usar ssh (el shell seguro) en lugar de rsh, basta con que cree la variable de entorno CVS_RSH de la forma apropiada:

```
floss$ CVS_RSH=ssh; export CVS_RSH
```

No se eche a temblar por el hecho de que el nombre de la variable es CVS_RSH y Ud. le está dando el valor ssh. Hay razones históricas para esto (la socorrida excusa Unix para todo, lo sé...). CVS_RSH puede apuntar al nombre de cualquier programa capaz de conectarle a un servidor remoto, ejecutar comandos y recibir su salida. Después de rsh, ssh es posiblemente el programa más común que cumple estos requisitos, aunque probablemente existan otros. Tenga en cuenta que este programa no debe modificar su flujo de información de ninguna manera. Esto deja fuera al rsh de Windows NT, puesto que convierte (o intenta convertir) entre las convenciones de fin de línea de DOS y Unix. En su caso, deberá conseguir algún otro rsh para Windows, o bien utilizar un método de acceso distinto.

Los métodos gserver y kserver no se utilizan tanto como los demás y no se cubren aquí. Son bastante parecidos a lo que aquí se ha cubierto hasta ahora; para más información sobre ellos, lea el Cederqvist.

Si sólo utiliza un repositorio y no quiere estar constantemente tecleando "-d repositorio", sólo tiene que crear la variable de entorno CVSROOT (que quizás debería haber sido llamada CVSREPOS, pero ya es demasiado tarde para eso):

```
floss$ CVSROOT=/usr/local/cvs
floss$ export CVSROOT
floss$ echo $CVSROOT
/usr/local/cvs
floss$
```

o quizás

```
floss$ CVSROOT=:pserver:jluis@cvs.foobar.com:/usr/local/cvs
floss$ export CVSROOT
floss$ echo $CVSROOT
:pserver:jluis@cvs.foobar.com:/usr/local/cvs
floss$
```

El resto de esta guía asume que ya ha creado la variable CVSROOT apuntando a su repositorio favorito, así que los ejemplos no mostrarán la opción -d. Si necesita acceder a muchos repositorios distintos, no debería crear la variable CVSROOT, sino limitarse a usar "-d repositorio" para indicar el repositorio a utilizar.

Comenzar un nuevo Proyecto

Si está estudiando el manejo de CVS para trabajar en un proyecto que ya se encuentra bajo control de CVS (es decir, que se guarda en un repositorio en alguna parte), probablemente querrá saltarse esta parte e ir directamente a la siguiente, "Obtener una copia de trabajo". Sin embargo, si lo que desea es tomar un código fuente ya existente y ponerlo bajo CVS, esta sección es para usted. Fíjese en que, a pesar de todo, se asume que dispone de acceso a un repositorio; vea ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ si necesita configurar un repositorio.

Introducir un nuevo proyecto en un repositorio de CVS es lo que se conoce como *importar*. El comando CVS a utilizar, como quizás ya haya adivinado, es

```
floss$ cvs import
```

excepto que le hacen falta más opciones (y debe estar en el sitio apropiado) para que el comando tenga éxito. Primero, vaya al directorio raíz de su árbol de proyectos:

```
floss$ cd miproyecto
floss$ ls
README.txt  a-subdir/   b-subdir/   hello.c
floss$
```

Este proyecto tiene dos ficheros (README.txt y hello.c) en el nivel más alto, además de dos subdirectorios (a-subdir y b-subdir), y algunos ficheros más (no mostrados en el ejemplo) dentro de esos subdirectorios. Al importar un proyecto, CVS importa todo lo que hay en el árbol, empezando por el directorio actual y yendo a continuación a todos los directorios que haya por debajo de éste. Por tanto, debería asegurarse de que sólo los ficheros que hay en este momento en el árbol son los que desea que formen parte del proyecto de forma permanente. Cualquier vieja copia de seguridad, borrador, etc., deberían ser eliminados antes.

La sintaxis general del comando de importación es

```
floss$ cvs import -m "mensaje" miproyecto marca_suministrador marca_lanzamiento
```

La opción -m (de "mensaje") sirve para especificar un breve informe que describe la importación. Éste será el primer informe de cambios que afecta al proyecto en todo su conjunto; cada envío realizado en el futuro tendrá su propio informe de cambios. Estos informes son obligatorios; si no se utiliza la opción -m, CVS lanzará automáticamente un editor (consultando previamente la variable de entorno EDITOR) para que escriba en él el informe a utilizar. Una vez que guarde el informe en disco y salga del editor, el proceso de importación seguirá adelante.

El siguiente argumento es el nombre del proyecto (usaremos "miproyecto"). Éste es el nombre con el cual podrá obtener copias desde el repositorio. (Lo que realmente sucede es que se crea un directorio con ese nombre en el repositorio; encontrará más información al respecto en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩.) El nombre que elija no tiene por qué ser igual al del directorio actual, aunque en la mayoría de los casos lo será.

Los argumentos marca_suministrador y marca_lanzamiento son información de registro adicional para CVS. No se preocupe de ellos ahora; poco importa lo que utilice en este momento. En ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩ podrá ver las raras ocasiones en las que son significativos; por ahora, utilizaremos un nombre de usuario y "start" respectivamente para estos dos argumentos.

Así pues, ya estamos listos para utilizar import:

```
floss$ cvs import -m "importación inicial a CVS" miproyecto jluis start
N miproyecto/hello.c
N miproyecto/README.txt
cvs import: Importing /usr/local/cvs/miproyecto/a-subdir
N miproyecto/a-subdir/loquesea.c
cvs import: Importing /usr/local/cvs/miproyecto/a-subdir/subsubdir
N miproyecto/a-subdir/subsubdir/fish.c
```

```
cvs import: Importing /usr/local/cvs/miproyecto/b-subdir
N miproyecto/b-subdir/random.c

No conflicts created by this import
floss$
```

Enhorabuena! Si ha ejecutado este comando (o algo similar), ya ha hecho por fin algo que afecta al repositorio.

Observando la salida del comando import, se dará cuenta de que CVS precede cada nombre de fichero con una letra, en este caso la "N" para indicar que se trata de un nuevo fichero. El uso de una letra a la izquierda para indicar el estado de un fichero es algo común en la salida de los comandos de CVS, tal como veremos más adelante también con los comandos "update" y "checkout".

Llegados a este punto, podría pensar que, puesto que ha importado el proyecto, puede empezar a trabajar en el árbol inmediatamente. Éste, sin embargo, no es el caso: el árbol de directorios actual no es todavía una copia de trabajo de CVS. Fue el origen para el comando de importación, cierto, pero no por éso se ha convertido por arte de magia en una copia de trabajo de CVS: para obtener una copia en la que poder trabajar, deberá tomarla del repositorio.

Pero antes, sin embargo, quizás quiera archivar el árbol de directorios actual. El motivo es que, una vez que las fuentes están en CVS, no querrá liarse y editar por error copias que no están bajo control de versión (puesto que esos cambios no se convertirán en parte del historial del proyecto). De ahora en adelante querrá hacer todas las ediciones sobre la copia de trabajo. Por otra parte, no le interesará eliminar completamente el árbol que ha importado, puesto que no ha verificado todavía si el repositorio dispone realmente de todos los ficheros. Por supuesto, puede estar un 99.999% seguro de que es así dado que el comando de importación no devolvió ningún error, pero, por qué correr riesgos? A menudo vale la pena ser paranoico, como puede confirmarle cualquier programador. Así que haga algo como esto:

```
floss$ ls
README.txt  a-subdir/   b-subdir/   hello.c
floss$ cd ..
floss$ ls
miproyecto/
floss$ mv miproyecto era_miproyecto
floss$ ls
era_miproyecto/
floss$
```

Hecho. Ahora sigue teniendo los ficheros originales, pero están claramente marcados como correspondientes a una versión obsoleta, así que no estarán ahí estorbándole cuando obtenga una verdadera copia de trabajo. Ahora sí, por fin, está listo para obtenerla.

Obtener una copia de trabajo

El comando para obtener un proyecto es exactamente el que cree que es:

```
floss$ cvs checkout miproyecto
cvs checkout: Updating miproyecto
```

```
U miproyecto/README.txt
U miproyecto/hello.c
cvs checkout: Updating miproyecto/a-subdir
U miproyecto/a-subdir/loquesea.c
cvs checkout: Updating miproyecto/a-subdir/subsubdir
U miproyecto/a-subdir/subsubdir/fish.c
cvs checkout: Updating miproyecto/b-subdir
U miproyecto/b-subdir/random.c

floss$ ls
miproyecto/      era_miproyecto/
floss$ cd miproyecto
floss$ ls
CVS/        README.txt  a-subdir/   b-subdir/   hello.c
floss$
```

Guau... su primera copia de trabajo! Su contenido es exactamente el mismo que el que ha importado, con el añadido de un subdirectorio llamado "CVS". Ahí es donde CVS guarda la información de control de versiones. De hecho, cada directorio presente en el proyecto tiene un subdirectorio CVS:

```
floss$ ls a-subdir
CVS/        subsubdir/  loquesea.c
floss$ ls a-subdir/subsubdir/
CVS/    fish.c
floss$ ls b-subdir
CVS/      random.c
```

El hecho de que CVS guarde la información de revisiones en subdirectorios llamados CVS conlleva que su proyecto no puede tener nunca subdirectorios propios llamados CVS. En la práctica, nunca he oído que esto supusiese un problema.

Antes de editar ningún fichero, echemos una ojeada a la caja negra:

```
floss$ cd CVS
floss$ ls
Entries     Repository  Root
floss$ cat Root
/usr/local/cvs
floss$ cat Repository
miproyecto
floss$
```

Nada misterioso por aquí. El fichero Root apunta al repositorio, y el fichero Repository apunta a un proyecto dentro del repositorio. Si esto le parece un poco confuso, permítame que se lo explique.

Hay una confusión muy extendida sobre la terminología empleada en CVS, y es que la palabra "repositorio" se utiliza para hacer referencia a dos cosas distintas. A veces, se utiliza para aludir al directorio raíz del repositorio (por ejemplo, /usr/local/cvs), que puede contener muchos proyectos; esto es a lo que se refiere el fichero Root. Pero, otras veces, se refiere a un subdirectorio particular específico de un proyecto dentro de la raíz de un repositorio (por ejemplo /usr/local/cvs/miproyecto, /usr/local/cvs/tuproyecto, o /usr/local/cvs/fish). El fichero "Repository" dentro de un subdirectorio CVS toma el segundo significado.

En este libro, "repositorio" generalmente significa "raíz", es decir, el repositorio situado en el nivel más alto, si bien, ocasionalmente, se usará para hacer referencia a un subdirectorio específico de un proyecto. Si el sentido que se le intenta dar no queda claro dentro del contexto, habrá texto que lo clarifique. Tenga en cuenta que el fichero "Repository" puede a veces contener la senda absoluta al nombre del proyecto en lugar de una senda relativa. Esto hace un tanto redundante al fichero Root:

```
floss$ cd CVS
floss$ cat Root
:pserver:jluis@cvs.foobar.com:/usr/local/cvs
floss$ cat Repository
/usr/local/cvs/miproyecto
floss$
```

El fichero "Entries" contiene información sobre cada uno de los ficheros que forman parte del proyecto. Cada línea se corresponde con un fichero, y sólo hay líneas para ficheros o subdirectorios en el directorio padre inmediato. Éste es el fichero CVS/Entries del directorio raíz de miproyecto:

```
floss$ cat Entries
/README.txt/1.1.1.1/Sun Apr 18 18:18:22 1999//
/hello.c/1.1.1.1/Sun Apr 18 18:18:22 1999//
D/a-subdir////
D/b-subdir////
```

El formato de cada línea es

```
/nombre de fichero/número de revisión/fecha de última modificación//
```

y las líneas de directorios vienen precedidas de la letra "D". En realidad, CVS no guarda un historial de los cambios realizados en los directorios, así que los campos de número de revisión y fecha están vacíos.

Las marcas de fecha guardan la fecha y hora de la última actualización (en horario universal, no local) de los ficheros presentes en la copia de trabajo. De esta forma, CVS puede saber fácilmente si un fichero ha sido modificado desde la última obtención de copia de trabajo, actualización o envío. Si la fecha registrada por el sistema de ficheros difiere de la fecha anotada en el fichero CVS/Entries, CVS sabe (sin ni siquiera tener que consultar el repositorio) que el fichero ha sido probablemente modificado.

Si examina los ficheros CVS/* en uno de los subdirectorios

```
floss$ cd a-subdir/CVS
floss$ cat Root
/usr/local/cvs
floss$ cat Repository
miproyecto/a-subdir
floss$ cat Entries
/loquesea.c/1.1.1.1/Sun Apr 18 18:18:22 1999//
D/subsubdir////
floss$
```

verá que el repositorio raíz no ha cambiado, pero el fichero "Repository" indica la situación de este subdirectorio dentro del proyecto, y el fichero "Entries" contiene líneas distintas.

Inmediatamente después de hacer una importación, el número de revisión de cada fichero en el proyecto se muestra como 1.1.1.1. Este número inicial de revisión es un caso un poco especial, así que no lo veremos en detalle aún; echaremos un vistazo más de cerca a los números de revisión una vez que hayamos hecho algunos cambios.

Versión vs. Revisión

El número interno de revisión que CVS guarda para cada fichero no tiene nada que ver con el número de versión del producto del que los ficheros forman parte. Por ejemplo, puede tener un proyecto formado por tres ficheros, cuyos números internos de revisión a 3 de mayo de 1999 eran 1.2, 1.7 y 2.48. Ese mismo día, empaqueta una nueva distribución del programa y la distribuye como ChachiSoft Versión 3. Esto es una decisión puramente de marketing y no afecta para nada a las revisiones de CVS. Los números de revisión de CVS son invisibles para sus clientes (a menos que les dé acceso al repositorio); el único número visible al público es el "3" de Versión 3. Por lo que respecta a CVS, podría haberlo llamado Versión 1729 si lo desease; el número de versión (o de distribución) no tiene nada que ver con el seguimiento interno de cambios que realiza CVS.

Para evitar confusiones, emplearé la palabra "revisión" para referirme solamente a los números de revisión interna de los ficheros controlados por CVS. A pesar de ello, me tomaré la libertad de llamar a CVS un "sistema de control de versiones", puesto que "sistema de control de revisiones" suena demasiado pedante.

Hacer un cambio

El proyecto, tal y como está en estos momentos, no hace mucho. Aquí están los contenidos de hello.c:

```
floss$ cat hello.c
#include <stdio.h>

void
main ()
{
   printf ("Hola, mundo!\n");
}
```

Ea, vamos a realizar nuestro primer cambio al proyecto desde que lo importamos; vamos a añadir la línea

```
printf ("Adiós, mundo!\n");
```

justo después de "Hola, mundo!". Llame a su editor favorito y haga el cambio:

```
floss$ emacs hello.c
  ...
```

Éste ha sido un cambio relativamente sencillo, en el que es poco posible que olvide lo que hizo. Pero en un proyecto más amplio, y más complejo, es muy posible que se ponga a editar un fichero, sea interrumpido por otra cosa, y, cuando vuelva varios días más tarde, sea incapaz de recordar exactamente lo que hizo, o incluso si cambió algo en absoluto. Lo cual nos trae a nuestra primera situación "CVS salva su vida": comparar su copia de trabajo con la que se encuentra en el repositorio.

Ver lo que Ud. (y otros) han hecho - comandos Update y Diff

Hasta ahora nos hemos referido a la "actualización" como una forma de traer a su copia de trabajo los cambios que se han realizado en el repositorio, es decir, como una manera de obtener los cambios hechos por otras personas. Sin embargo, la actualización es un proceso algo más complejo que esto, puesto que compara el estado global de su copia de trabajo con el estado del proyecto que se encuentra en el repositorio. Incluso si nada ha cambiado en el repositorio desde que se obtuvo la copia, puede que algo en la copia de trabajo sí lo haya hecho, y "update" también le mostrará esto:

```
floss$ cvs update
cvs update: Updating .
M hello.c
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
```

La "M" al lado de hello.c significa que el fichero ha sido modificado desde que se obtuvo la copia, y que las modificaciones no se han enviado aún al repositorio.

A veces, todo lo que necesita es simplemente saber qué ficheros ha editado. Sin embargo, si desea echar un vistazo más de cerca a los cambios, puede solicitar un informe detallado en formato diff. El comando diff compara los ficheros que puedan haberse modificado en la copia de trabajo con sus homónimos en el repositorio, mostrando a continuación cualquier posible diferencia:

```
floss$ cvs diff
cvs diff: Diffing .
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 hello.c
6a7
>   printf ("Adiós, mundo!\n");
cvs diff: Diffing a-subdir
cvs diff: Diffing a-subdir/subsubdir
cvs diff: Diffing b-subdir
```

Aunque un poco liosa, esta información es útil, si bien todavía hay un montón de ruido ahí. Para empezar, puede ignorar la mayor parte de las líneas del comienzo, dado que sólo hacen referencia al nombre del fichero del repositorio e indican el número de la última revisión enviada al mismo. Son datos útiles en otras circunstancias (las veremos en detalle más adelante), pero no las necesita cuando sólo quiere hacerse una idea de los cambios que se han hecho en la copia de trabajo.

Una molestia más seria a la hora de leer el diff es que CVS anuncia su entrada en escena a medida que va entrando en cada directorio durante la actualización. Esto puede ser útil durante largas actualizaciones en grandes proyectos, puesto que le da una idea del tiempo que va a necesitar el comando, pero ahora mismo lo único que hace es molestarle mientras intenta interpretar lo que está leyendo. Así pues, digámosle a CVS que no diga nada sobre su trabajo, con la opción global -Q (de "Que te calles", obviamente):

```
floss$ cvs -Q diff
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 hello.c
6a7
>   printf ("Adiós, mundo!\n");
```

Mejor - al menos, parte del ruido ha desaparecido. Sin embargo, el diff es aún difícil de leer. Le está diciendo que en la línea 6 se añadió una línea nueva (que se convirtió en la línea 7) cuyo contenido es:

```
printf ("Adiós, mundo!\n");
```

El signo ">" que precede a la línea en el diff le dice que esta línea está presente en la nueva versión del fichero, pero no en la antigua.

Sin embargo, el formato podría ser aún más legible. Muchas personas encuentran el formato "de contexto" de diff más fácil de leer, porque muestra menos líneas de contexto delimitando los cambios. Los diffs de contexto pueden generarse pasando a diff la opción -c:

```
floss$ cvs -Q diff -c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.1.1.1
diff -c -r1.1.1.1 hello.c
*** hello.c     1999/04/18 18:18:22     1.1.1.1
--- hello.c     1999/04/19 02:17:07
***************
*** 4,7 ****
---4,8 --
  main ()
  {
    printf ("Hola, mundo!\n");
+   printf ("Adiós, mundo!\n");
  }
```

Esto sí está claro! Incluso si no está acostumbrado a leer diffs de contexto, un vistazo a esta información mostrará de forma bastante obvia lo que ha sucedido: se añadió una nueva línea (el + en la primera columna significa que se ha añadido una línea) entre la línea que imprime "Hola, mundo!" y la llave final.

No es necesario que seamos capaces de interpretar perfectamente los diffs de contexto (esto es trabajo para el programa "patch"), pero le será útil tomarse un tiempo para adquirir al menos una cierta familiaridad con el formato. Las primeras dos líneas (después del rollo del comienzo) son

```
*** hello.c     1999/04/18 18:18:22     1.1.1.1
--- hello.c     1999/04/19 02:17:07
```

y le dicen qué se está comparando con qué. En este caso, la revisión 1.1.1.1 de hello.c se está comparando con una versión modificada del mismo fichero (motivo por el cual no

aparece número alguno de revisión en la segunda línea, porque los cambios de la copia de trabajo no se han enviado todavía al repositorio). Las líneas de asteriscos y guiones identifican secciones situadas más adelante en el fichero de diferencias. Más adelante, una línea de asteriscos seguida de una franja de valores precede a una sección del fichero original. Después, una línea de guiones, con una franja de números de línea nuevos y potencialmente distintos, precede a una sección del fichero modificado. Estas secciones están organizadas por pares: por un lado la parte del fichero antiguo, y por otro lado la parte del fichero nuevo.

Nuestro fichero de diferencias tiene uno de estos pares:

```
***************
*** 4,7 ****
--- 4,8 --
  main ()
  {
    printf ("Hola, mundo!\n");
+   printf ("Adiós, mundo!\n");
  }
```

La primera sección del par está vacía, lo que significa que no se ha eliminado nada del fichero original. La segunda sección indica que, en el lugar correspondiente del nuevo fichero, se ha añadido una nueva línea, que aparece marcada con un signo "+". (Cuando el diff cita partes de ficheros, se reserva las primeras dos columnas a la izquierda para códigos especiales, como el "+", así que el trozo entero aparenta estar justificado a la izquierda con dos espacios. Esta justificación extra desaparece, por supuesto, cuando se aplica el fichero de diferencias.)

La franja de números de líneas muestra el alcance del par de diferencias, incluyendo líneas de contexto. En el fichero original, el par estaba en las líneas 4 a la 7; en el nuevo fichero, son las líneas 4 a la 8 debido a la nueva línea que se ha añadido. Fíjese en que el fichero de diferencias no necesita enseñar ninguna cosa del fichero original puesto que no se ha eliminado nada; sólo nos ha mostrado la franja afectada y ha continuación ha saltado a la segunda sección del par de diferencias.

Aquí hay otro diff de contexto, procedente esta vez de un proyecto real mío:

```
floss$ cvs -Q diff -c
Index: cvs2cl.pl
===================================================================
RCS file: /usr/local/cvs/kfogel/code/cvs2cl/cvs2cl.pl,v
retrieving revision 1.76
diff -c -r1.76 cvs2cl.pl
*** cvs2cl.pl   1999/04/13 22:29:44     1.76
--- cvs2cl.pl   1999/04/19 05:41:37
***************
*** 212,218 ****
          # can contain uppercase and lowercase letters, digits, '-',
          # and '_'. However, it's not our place to enforce that, so
          # we'll allow anything CVS hands us to be a tag:
!         /^\s([^:]+): ([0-9.]+)$/;
          push (@{$symbolic_names{$2}}, $1);
        }
```

```
      }
-- 212,218 --
          # can contain uppercase and lowercase letters, digits, '-',
          # and '_'. However, it's not our place to enforce that, so
          # we'll allow anything CVS hands us to be a tag:
!         /^\s([^:]+): ([\d.]+)$/;
          push (@{$symbolic_names{$2}}, $1);
        }
      }
```

El signo de exclamación indica que la línea marcada difiere del fichero antiguo al nuevo. Dado que no hay ningún signo "+" o "-", sabemos que el número total de líneas del fichero sigue siendo el mismo.

Éstas son otras diferencias de contexto del mismo proyecto, esta vez un poco más complejas:

```
floss$ cvs -Q diff -c
Index: cvs2cl.pl
===================================================================
RCS file: /usr/local/cvs/kfogel/code/cvs2cl/cvs2cl.pl,v
retrieving revision 1.76
diff -c -r1.76 cvs2cl.pl
*** cvs2cl.pl   1999/04/13 22:29:44     1.76
--- cvs2cl.pl   1999/04/19 05:58:51
***************
*** 207,217 ****
}
        else    # we're looking at a tag name, so parse & store it
        {
-         # According to the Cederqvist manual, in node "Tags", "Tag
-         # names must start with an uppercase or lowercase letter and
-         # can contain uppercase and lowercase letters, digits, '-',
-         # and '_'. However, it's not our place to enforce that, so
-         # we'll allow anything CVS hands us to be a tag:
          /^\s([^:]+): ([0-9.]+)$/;
          push (@{$symbolic_names{$2}}, $1);
        }
- 207,212 --
***************
*** 223,228 ****
--- 218,225 --
      if (/^revision (\d\.[0-9.]+)$/) {
        $revision = "$1";
      }
+
+     # Esta línea ha sido añadida, lo admito, sólo para este ejemplo de diff.

      # If have file name but not time and author, and see date or
      # author, then grab them:
```

Este fichero diff tiene dos pares de diferencias. En el primero se han eliminado cinco líneas (estas líneas se muestran sólo en la primera sección del par, y la cuenta de líneas de la segunda sección indica que tiene menos líneas). Una línea continua de asteriscos hace las veces de delimitador entre pares, y en el segundo par vemos que se han añadido dos líneas: una línea en blanco y un comentario inútil. Observe cómo los números de línea compensan el efecto del par anterior. En el fichero original, la franja del segundo par iba desde 223 hasta 228; en el nuevo fichero, dado que la eliminación tuvo lugar en el primer par, la franja de líneas abarca desde la 218 hasta la 225.

Enhorabuena, en este momento tiene posiblemente toda la experiencia que necesita para poder interpretar ficheros de diferencias.

CVS y argumentos implícitos

En cada uno de los comandos de CVS explicados hasta el momento, quizás se haya dado cuenta de que no hemos indicado fichero alguno en la línea de comando. Por ejemplo, hemos utilizado

```
floss$ cvs diff
```

en lugar de

```
floss$ cvs diff hello.c
```

y

```
floss$ cvs update
```

en lugar de

```
floss$ cvs update hello.c
```

El principio que dicta esto es que si usted no indica ningún nombre de fichero, CVS actuará sobre todos los ficheros sobre los que el comando resultaría apropiado. Esto incluye también ficheros en subdirectorios situados por debajo del directorio actual; CVS automáticamente desciende desde el directorio actual hasta cada subdirectorio en el árbol. Por ejemplo, si modificó b-subdir/random.c y a-subdir/subsubdir/fish.c, invocar una actualización podría devolver unos resultados como éstos:

```
floss$ cvs update
cvs update: Updating .
M hello.c
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
M a-subdir/subsubdir/fish.c
cvs update: Updating b-subdir
M b-subdir/random.c
floss$
```

o mejor aún:

```
floss$ cvs -q update
M hello.c
M a-subdir/subsubdir/fish.c
M b-subdir/random.c
floss$
```

Nota: La opción -q es una versión "ligera" de -Q. Si se hubiera usado -Q, el comando no habría mostrado nada en absoluto, porque las notas de modificación son consideradas

mensajes de información no esenciales. La versión en minúsculas, -q, es menos estricta: suprime los mensajes que posiblemente no queramos, a la vez que permite que otros mensajes, posiblemente más útiles, sí lleguen hasta nosotros.

También puede nombrar ficheros específicos que desee actualizar:

```
floss$ cvs update hello.c b-subdir/random.c
M hello.c
M b-subdir/random.c
floss$
```

y CVS sólo examinará estos ficheros, ignorando cualquier otro.

Lo cierto es que es más habitual no limitar la actualización a determinados ficheros, puesto que en la mayoría de los casos querrá actualizar el árbol de directorios al completo de una sola vez. En todo caso, recuerde que las actualizaciones que hacemos aquí a modo de ejemplo sólo muestran que algunos ficheros se han modificado de forma local, porque aún no se ha cambiado nada en el repositorio. Cuando otras personas están trabajando en el mismo proyecto que usted, siempre existe la posibilidad de que la actualización incorpore algunos cambios nuevos del repositorio en sus ficheros locales, en cuyo caso sí puede resultarle útil indicar los ficheros en concreto que desea actualizar.

El mismo principio puede aplicarse a otros comandos de CVS. Por ejemplo, con diff, puede ver los cambios habidos en un fichero cada vez:

```
floss$ cvs diff -c b-subdir/random.c
Index: b-subdir/random.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.1.1.1
diff -c -r1.1.1.1 random.c
*** b-subdir/random.c   1999/04/18 18:18:22     1.1.1.1
--- b-subdir/random.c   1999/04/19 06:09:48
***************
*** 1 ****
! /* Un fichero en C completamente vacío. */
--- 1,8 --
! /* Imprimir un número aleatorio. */
!
! #include <stdio.h>
!
! void main ()
! {
!   printf ("un número aleatorio\n");
! }
```

o ver todos los cambios de una sola vez (agárrese al asiento, esto va a ser un diff bastante grande):

```
floss$ cvs -Q diff -c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.1.1.1
diff -c -r1.1.1.1 hello.c
```

```
*** hello.c     1999/04/18 18:18:22     1.1.1.1
--- hello.c     1999/04/19 02:17:07
***************
*** 4,7 ****
--- 4,8 --
  main ()
  {
    printf ("Hola, mundo!\n");
+   printf ("Adiós, mundo!\n");
  }
Index: a-subdir/subsubdir/fish.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/a-subdir/subsubdir/fish.c,v
retrieving revision 1.1.1.1
diff -c -r1.1.1.1 fish.c
*** a-subdir/subsubdir/fish.c   1999/04/18 18:18:22     1.1.1.1
--- a-subdir/subsubdir/fish.c   1999/04/19 06:08:50
***************
*** 1 ****
! /* Un fichero en C completamente vacío. */
--- 1,8 --
! #include <stdio.h>
!
! void main ()
! {
!   while (1) {
!     printf ("fish\n");
!   }
! }
Index: b-subdir/random.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.1.1.1
diff -c -r1.1.1.1 random.c
*** b-subdir/random.c   1999/04/18 18:18:22     1.1.1.1
--- b-subdir/random.c   1999/04/19 06:09:48
***************
*** 1 ****
! /* Un fichero en C completamente vacío. */
--- 1,8 --
! /* Imprimir un número aleatorio. */
!
! #include <stdio.h>
!
! void main ()
! {
!   printf ("un número aleatorio\n");
! }
```

En cualquier caso, como puede ver en estos diffs, queda claro que el proyecto está listo para debutar. Enviemos los cambios al repositorio.

Enviar cambios al repositorio

El comando *commit* envía las modificaciones al repositorio. Si no indica ningún fichero, el comando enviará todos los cambios al repositorio; sin embargo, si así lo prefiere, puede indicarle el nombre de uno o más ficheros concretos a enviar, en cuyo caso el resto de los ficheros serán ignorados.

Aquí enviamos un fichero nombrándolo expresamente, y otros dos por alusiones:

```
floss$ cvs commit -m "ahora también dice adiós" hello.c
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <--  hello.c
new revision: 1.2; previous revision: 1.1
done
floss$ cvs commit -m "añadido código C"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in a-subdir/subsubdir/fish.c;
/usr/local/cvs/miproyecto/a-subdir/subsubdir/fish.c,v  <--  fish.c
new revision: 1.2; previous revision: 1.1
done
Checking in b-subdir/random.c;
/usr/local/cvs/miproyecto/b-subdir/random.c,v  <--  random.c
new revision: 1.2; previous revision: 1.1
done
floss$
```

Deténgase un momento a leer la salida detenidamente. La mayor parte de lo que dice se explica por sí solo. Una cosa de la que tal vez se dé cuenta es que los números de revisión se han incrementado (como era de esperar), pero las revisiones originales se listan como 1.1 en lugar de 1.1.1.1 como vimos anteriormente para el caso del fichero Entries.

Hay una explicación para esta discrepancia, pero no es muy importante, y tiene que ver con un significado especial que CVS da a la revisión 1.1.1.1: en la mayor parte de los casos, podemos decir simplemente que los ficheros reciben un número de revisión 1.1 al hacer una importación, pero, -por motivos que sólo CVS conoce- el número aparece como 1.1.1.1 en el fichero Entries hasta que el fichero es enviado al repositorio por primera vez.

Números de revisión

Cada fichero en un proyecto tiene su propio número de revisión. Cuando un fichero es enviado al repositorio, la última parte del número de revisión se incrementa en una unidad. Por tanto, los diferentes ficheros que forman parte de un proyecto pueden tener siempre números de revisión (a veces muy) diferentes. Esto sólo significa que algunos ficheros han sido modificados (e incorporados en el repositorio) con más frecuencia que otros.

En este momento quizás se pregunte qué sentido tiene la parte situada a la izquierda del punto decimal, cuando la única parte que cambia es la situada a la derecha. Pues bien,

a pesar de que CVS nunca incrementa automáticamente el número situado a la izquierda, este número puede ser incrementado a petición del usuario. Esto es algo que se usa en muy contadas ocasiones, y no lo cubriremos en esta guía.

Volviendo al tema, en el proyecto de ejemplo que hemos estado usando, acabábamos de enviar al repositorio los cambios que habíamos realizado en tres ficheros. Cada uno de estos ficheros es ahora la revisión 1.2, pero el resto de ficheros del proyecto son aún la revisión 1.1. Cuando usted solicita al repositorio una copia de un proyecto, siempre obtiene la última revisión de cada fichero allí presente. Esto es lo que el usuario mperez vería si ahora mismo solicitase una copia de miproyecto y observase los números de revisión del directorio raíz:

```
paste$ cvs -q -d :pserver:mperez@cvs.foobar.com:/usr/local/cvs co miproyecto
U miproyecto/README.txt
U miproyecto/hello.c
U miproyecto/a-subdir/loquesea.c
U miproyecto/a-subdir/subsubdir/fish.c
U miproyecto/b-subdir/random.c
paste$ cd miproyecto/CVS
paste$ cat Entries
/README.txt/1.1.1.1/Sun Apr 18 18:18:22 1999//
/hello.c/1.2/Mon Apr 19 06:35:15 1999//
D/a-subdir////
D/b-subdir////
paste$
```

El fichero hello.c (entre otros) se encuentra ahora en su revisión 1.2, mientras que el fichero README.txt está aún en la revisión inicial (1.1.1.1, también conocida como 1.1).

Si mperez añade ahora la línea

```
printf ("entre hola y adiós\n");
```

a hello.c y lo envía, el número de revisión del fichero se incrementará una vez más:

```
paste$ cvs ci -m "añadida una nueva línea entremedias"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <--  hello.c
new revision: 1.3; previous revision: 1.2
done
paste$
```

Ahora hello.c está en la revisión 1.3, fish.c y random.c están aún en la revisión 1.2, y los demás ficheros en la revisión 1.1.

Observe que el comando fue dado como cvs ci en lugar de cvs commit. La mayor parte de los comandos CVS tienen una forma abreviada, para hacer más fácil el escribirlos. Para checkout, update y commit, las versiones abreviadas son co, up y ci, respectivamente. Puede obtener una lista de todas las formas abreviadas ejecutando el comando `cvs --help-synonyms`.

Normalmente puede ignorar el número de revisión de un fichero. En la mayoría de los casos, estos números son simplemente anotaciones internas que CVS gestiona

automáticamente. Sin embargo, ser capaz de encontrar y comparar números de revisión es algo muy útil cuando tiene que obtener (o establecer diferencias respeto a) una copia antigua del fichero.

Examinar el fichero Entries no es la única forma de descubrir un número de revisión. Puede usar también el comando status:

```
paste$ cvs status hello.c
===================================================================
File: hello.c           Status: Up-to-date

   Working revision:    1.3     Tue Apr 20 02:34:42 1999
   Repository revision: 1.3     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

el cual, cuando se invoca sin nombrar ningún fichero, muestra el estado de todos los ficheros que conforman el proyecto:

```
paste$ cvs status
cvs status: Examining.
===================================================================
File: README.txt        Status: Up-to-date

   Working revision:    1.1.1.1 Sun Apr 18 18:18:22 1999
   Repository revision: 1.1.1.1 /usr/local/cvs/miproyecto/README.txt,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: hello.c           Status: Up-to-date

   Working revision:    1.3     Tue Apr 20 02:34:42 1999
   Repository revision: 1.3     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

cvs status: Examining a-subdir
===================================================================
File: loquesea.c        Status: Up-to-date

   Working revision:    1.1.1.1 Sun Apr 18 18:18:22 1999
   Repository revision: 1.1.1.1 /usr/local/cvs/miproyecto/a-subdir/loquesea.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

cvs status: Examining a-subdir/subsubdir
===================================================================
```

```
File: fish.c            Status: Up-to-date

   Working revision:    1.2     Mon Apr 19 06:35:27 1999
   Repository revision: 1.2     /usr/local/cvs/miproyecto/
                                a-subdir/subsubdir/fish.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

cvs status: Examining b-subdir
===================================================================
File: random.c          Status: Up-to-date

   Working revision:    1.2     Mon Apr 19 06:35:27 1999
   Repository revision: 1.2     /usr/local/cvs/miproyecto/b-subdir/random.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

paste$
```

Limítese a ignorar las partes de la salida que no entienda; de hecho, éste es por regla general un buen consejo al utilizar CVS. A menudo, el pequeño trozo de información que está buscando vendrá acompañado de otra mucha información que no le interesa, y que quizás ni siquiera comprenda. Esta situación es normal; simplemente tome lo que necesite y olvídese de todo lo demás.

En el ejemplo anterior, las partes que nos interesan son las primeras tres líneas (sin contar la línea en blanco) de la información de estado de cada fichero. La primera línea es la más importante, puesto que le dice el nombre del fichero y su estado en la copia de trabajo. Todos los ficheros están en este momento sincronizados con el repositorio, así que todos dicen `Up-to-date`. Sin embargo, si random.c hubiera sido modificado y el cambio no se hubiese enviado al repositorio, podríamos encontrarnos algo como esto:

```
===================================================================
File: random.c          Status: Locally Modified

   Working revision:    1.2     Mon Apr 19 06:35:27 1999
   Repository revision: 1.2     /usr/local/cvs/miproyecto/b-subdir/random.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Los números de revisión de la copia de trabajo y de la copia presente en el repositorio le informan de si el fichero está o no sincronizado con la copia que hay en el repositorio. Volviendo a nuestra copia de trabajo original (la copia de jluis, que no ha visto todavía el cambio habido en hello.c), vemos lo siguiente:

```
floss$ cvs status hello.c
===================================================================
File: hello.c           Status: Needs Patch
```

```
   Working revision:    1.2     Mon Apr 19 02:17:07 1999
   Repository revision: 1.3     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

floss$
```

Esto nos dice que alguien ha efectuado cambios en hello.c, elevando a 1.3 el número de revisión de la copia que hay en el repositorio, y que esta copia de trabajo está aún en la revisión 1.2. La línea "Status: Needs Patch" significa que la siguiente actualización traerá los cambios del repositorio y los aplicará a la copia de trabajo del fichero.

Supongamos por un momento que ignoramos completamente el cambio que mperez ha hecho a hello.c, así que no utilizamos status ni update, sino que simplemente procedemos a editar nuestro fichero local, realizando un cambio ligeramente distinto en el mismo punto del fichero. Esto nos lleva a nuestro primer conflicto.

Detección y resolución de conflictos

Detectar un conflicto es bastante sencillo. Al invocar una actualización, CVS le dice, bien a las claras, que existe un conflicto. Pero primero, creemos el conflicto en sí: editemos el fichero hello.c para insertar la línea

```
printf ("este cambio generará un conflicto\n");
```

exactamente donde mperez introdujo esto:

```
printf ("entre hola y adiós\n");
```

En este momento, el estado de nuestra copia de hello.c es

```
floss$ cvs status hello.c
===================================================================
File: hello.c           Status: Needs Merge

   Working revision:    1.2     Mon Apr 19 02:17:07 1999
   Repository revision: 1.3     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

floss$
```

lo que significa que ha habido cambios tanto en la copia del repositorio como en nuestra copia de trabajo, y que estos cambios necesitan ser fusionados (CVS no es aún consciente de que los cambios entrarán en conflicto, porque aún no hemos intentado hacer una actualización). Cuando hagamos una actualización, veremos esto:

```
floss$ cvs update hello.c
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into hello.c
rcsmerge: warning: conflicts during merge
```

```
cvs update: conflicts found in hello.c
C hello.c
floss$
```

La última línea es la clave. La "C" situada a la izquierda del nombre del fichero indica que los cambios han sido fusionados [en nuestra copia de trabajo], pero que entran en conflicto. El contenido de hello.c muestra ahora ambos cambios:

```
#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
<<<<<<< hello.c
  printf ("este cambio generará un conflicto\n");
=======
  printf ("entre hola y adiós\n");
>>>>>>> 1.3
  printf ("Adiós, mundo!\n");
}
```

Los conflictos siempre se muestran delimitados por marcadores de conflicto, en el siguiente formato:

```
<<<<<<< (nombre de fichero)
  (cambios en la copia de trabajo, todavía no enviados al repositorio)
  blah blah blah
=======
  (cambios procedentes del repositorio)
  blah blah blah
  etc.
>>>>>>> (último número de revisión en el repositorio)
```

El fichero Entries también muestra que el fichero se encuentra en este momento en un estado intermedio:

```
floss$ cat CVS/Entries
/README.txt/1.1.1.1/Sun Apr 18 18:18:22 1999//
D/a-subdir////
D/b-subdir////
/hello.c/1.3/Result of merge+Tue Apr 20 03:59:09 1999//
floss$
```

La manera de resolver el conflicto es editar el fichero de forma que tenga el texto que resulte apropiado, eliminando de paso los marcadores de conflicto, y después enviarlo al repositorio. Esto no significa necesariamente elegir entre uno o otro cambio; podría decidir que ninguno de los dos cambios es satisfactorio y reescribir la sección donde aparece el conflicto, o incluso el fichero al completo. En este caso, vamos a favorecer el primero de los cambios, pero con una puntuación y uso de las mayúsculas ligeramente distintos de los empleados por mperez:

```
floss$ emacs hello.c
  (editamos el fichero...)
floss$ cat hello.c
```

```
#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
  printf ("ENTRE HOLA Y ADIÓS.\n");
  printf ("Adiós, mundo!\n");
}
floss$ cvs ci -m "alterada la línea del medio"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <-  hello.c
new revision: 1.4; previous revision: 1.3
done
floss$
```

Averiguar quién hizo qué (leyendo informes de cambios)

A estas alturas, el proyecto ha pasado ya por varios cambios. Si está intentado hacerse una idea de lo sucedido hasta el momento, no querrá necesariamente examinar con detalle cada fichero de diferencias. Examinar los informes de cambios sería lo ideal, y puede hacerlo con el comando log:

```
floss$ cvs log
(páginas y páginas de salida omitidas)
```

La salida del comando tiende a ser bastante detallada. Veamos los informes de cambios para un único fichero:

```
floss$ cvs log hello.c
RCS file: /usr/local/cvs/miproyecto/hello.c,v
Working file: hello.c
head: 1.4
branch:
locks: strict
access list:
symbolic names:
        start: 1.1.1.1
        jluis: 1.1.1
keyword substitution: kv
total revisions: 5;     selected revisions: 5
description:
----------------------------
revision 1.4
date: 1999/04/20 04:14:37;  author: jluis;  state: Exp;  lines: +1 -1
alterada la línea del medio
----------------------------
```

```
revision 1.3
date: 1999/04/20 02:30:05;  author: mperez;  state: Exp;  lines: +1 -0
añadida una nueva línea entremedias
----------
revision 1.2
date: 1999/04/19 06:35:15;  author: jluis;  state: Exp;  lines: +1 -0
ahora también dice adiós
----------
revision 1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;
branches:  1.1.1;
Revisión inicial
----------
revision 1.1.1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;  lines: +0 -0
Importación inicial en CVS
=============================================================================
floss$
```

Como es habitual, hay mucha información al comienzo que puede simplemente ignorar. La parte que nos interesa viene después de cada línea de guiones, en un formato que se explica por sí solo.

Cuando se envían muchos ficheros a la vez al repositorio, todos tienen un mismo informe de cambios, algo que puede ser útil para seguir los cambios. Por ejemplo, recuerda cuando enviamos fish.c y random.c a la vez? Se hizo de esta forma:

```
floss$ cvs commit -m "añadido código C"
Checking in a-subdir/subsubdir/fish.c;
/usr/local/cvs/miproyecto/a-subdir/subsubdir/fish.c,v  <-  fish.c
new revision: 1.2; previous revision: 1.1
done
Checking in b-subdir/random.c;
/usr/local/cvs/miproyecto/b-subdir/random.c,v  <-  random.c
new revision: 1.2; previous revision: 1.1
done
floss$
```

El resultado de esta operación fue el envío de ambos ficheros con un mismo informe de cambios: "añadido código C". Se da el caso de que ambos ficheros empezaban en la revisión 1.1 y pasaron a la 1.2, pero esto es sólo una coincidencia; si random.c estuviera en la revisión 1.29, habría pasado a la 1.30 tras concluir este envío, y la revisión 1.30 tendría el mismo informe de cambios que la revisión 1.2 de fish.c.

Al utilizar cvs log con estos ficheros, verá el informe que ambos comparten:

```
floss$ cvs log a-subdir/subsubdir/fish.c b-subdir/random.c

RCS file: /usr/local/cvs/miproyecto/a-subdir/subsubdir/fish.c,v
Working file: a-subdir/subsubdir/fish.c
head: 1.2
branch:
locks: strict
```

```
access list:
symbolic names:
        start: 1.1.1.1
        jluis: 1.1.1
keyword substitution: kv
total revisions: 3;     selected revisions: 3
description:
----------
revision 1.2
date: 1999/04/19 06:35:27;  author: jluis;  state: Exp;  lines: +8 -1
añadido código C
----------
revision 1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;
branches:  1.1.1;
Revisión inicial
----------
revision 1.1.1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;  lines: +0 -0
Importación inicial en CVS
=============================================================================
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
Working file: b-subdir/random.c
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        start: 1.1.1.1
        jluis: 1.1.1
keyword substitution: kv
total revisions: 3;     selected revisions: 3
description:
----------
revision 1.2
date: 1999/04/19 06:35:27;  author: jluis;  state: Exp;  lines: +8 -1
añadido código C
----------
revision 1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;
branches:  1.1.1;
Revisión inicial
----------
revision 1.1.1.1
date: 1999/04/18 18:18:22;  author: jluis;  state: Exp;  lines: +0 -0
Importación inicial en CVS
=============================================================================
floss$
```

Por esta información sabrá que las dos revisiones fueron parte del mismo envío (el hecho de que las fechas de las dos revisiones sean las mismas, o muy próximas entre sí, es una prueba más).

Examinar los informes de cambios es una buena forma de hacerse rápidamente una idea de lo que ha estado sucediendo en un proyecto o de saber lo que pasó con un fichero específico en un momento determinado. Existen también muchas herramientas libres diseñadas para convertir la salida bruta de cvs log a formatos más concisos y legibles (por ejemplo, al estilo ChangeLog de GNU); no cubriremos estas herramientas en esta guía, pero serán presentadas en ⟨undefined⟩ [Herramientas de terceros], page ⟨undefined⟩.

Examinar y deshacer cambios

Supongamos que, mientras estamos leyendo los informes de cambios, mperez se percata de que jluis hizo el cambio más reciente a hello.c:

```
revision 1.4
date: 1999/04/20 04:14:37;  author: jluis;  state: Exp;  lines: +1 -1
alterada la línea del medio
```

y se pregunta qué hizo jluis exactamente. En términos formales, la pregunta que se hace mperez es, "Cuál es la diferencia entre mi revisión (1.3) de hello.c, y la revisión de jluis que vino a continuación (1.4)?" La forma de averiguarlo es utilizar el comando diff, pero esta vez comparando las dos últimas revisiones utilizando la opción de comando -r para especificar ambos números:

```
paste$ cvs diff -c -r 1.3 -r 1.4 hello.c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.3
retrieving revision 1.4
diff -c -r1.3 -r1.4
*** hello.c     1999/04/20 02:30:05     1.3
--- hello.c     1999/04/20 04:14:37     1.4
***************
*** 4,9 ****
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("entre hola y adiós\n");
    printf ("Adiós, mundo!\n");
  }
--- 4,9 --
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("ENTRE HOLA Y ADIÓS.\n");
    printf ("Adiós, mundo!\n");
  }
paste$
```

El cambio es bastante obvio, visto de esta forma. Dado que los números de revisión se dan en orden cronológico (por lo general, una buena idea), el comando los muestra de forma ordenada. Si sólo se da un número de revisión, CVS utiliza como segundo número la revisión actual del fichero presente en la copia de trabajo.

Cuando mperez ve estos cambios, decide al momento que le gusta más su forma de hacer las cosas, así que decide "deshacerlo", esto es, ir una revisión atrás.

Sin embargo, esto no significa que desee perder su revisión 1.4. Si bien en un sentido estrictamente técnico es probablemente posible conseguir este efecto en CVS, raramente existe razón alguna para hacerlo. Es mucho más deseable guardar la revisión 1.4 en el historial y hacer una nueva revisión 1.5 idéntica a la 1.3: de esta forma, la operación de deshacer se convierte en parte del historial del fichero.

La única pregunta es, cómo puede usted obtener el contenido de la revisión 1.3 y ponerlo en la 1.5?

En este caso en particular, dado que el cambio es muy sencillo, mperez puede seguramente limitarse a editar el fichero a mano para hacerlo idéntico a la revisión 1.3, y entonces enviarlo al repositorio. Sin embargo, si los cambios son más complejos (como suele ocurrir en un proyecto real), intentar recrear a mano la revisión antigua será irremediablemente algo donde se producirán errores. Por tanto, vamos a hacer que mperez utilice CVS para obtener y reenviar el contenido de la antigua revisión.

Hay dos formas igualmente buenas de conseguir esto: la lenta y dolorosa, y la rápida y vistosa. Vamos a examinar primero la lenta y dolorosa.

El método lento de deshacer cosas

Este método utiliza la combinación de opciones -p y -r al hacer la actualización. La opción -p envía el contenido de la revisión indicada a la salida estándar.

De por sí, no es que esto sea algo terriblemente útil, dado que el contenido del fichero desaparecerá rápidamente por la ventana, dejando intacta la copia de trabajo. Sin embargo, redirigiendo la salida estándar al fichero, el fichero terminará recibiendo el contenido de la antigua revisión. Sería lo mismo que haber editado el fichero a mano para llevarlo a ese estado.

Primero, sin embargo, mperez necesita ponerse al día respecto al repositorio:

```
paste$ cvs update
cvs update: Updating .
U hello.c
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
paste$ cat hello.c
#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
  printf ("ENTRE HOLA Y ADIÓS.\n");
```

```
  printf ("Adiós, mundo!\n");
}
paste$
```

Lo siguiente que hace es emplear la opción -p al invocar la actualización, para asegurarse de que la revisión 1.3 es justo la que quiere:

```
paste$ cvs update -p -r 1.3 hello.c
===================================================================
Checking out hello.c
RCS:  /usr/local/cvs/miproyecto/hello.c,v
VERS: 1.3
***************
#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
  printf ("entre hola y adiós\n");
  printf ("Adiós, mundo!\n");
}
```

Ooops, hay algunas líneas inútiles al comienzo. En realidad estas líneas no se envían a la salida estándar sino al flujo de error estándar, así que son inofensivas. Aún así, estas líneas hacen más difícil interpretar la información, y pueden ser suprimidas con -Q:

```
paste$ cvs -Q update -p -r 1.3 hello.c
#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
  printf ("entre hola y adiós\n");
  printf ("Adiós, mundo!\n");
}
paste$
```

Ahí estamos - esto es exactamente lo que mperez quería obtener. El siguiente paso es poner ese contenido en el fichero de la copia de trabajo, usando una redirección Unix (que es lo que hace el signo ">"):

```
paste$ cvs -Q update -p -r 1.3 hello.c > hello.c
paste$ cvs update
cvs update: Updating .
M hello.c
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
paste$
```

Al hacer ahora la actualización, el fichero aparece como modificado, lo cual tiene sentido dado que su contenido ha cambiado. Específicamente, tiene el mismo contenido que la

antigua revisión 1.3 (no es que CVS se dé cuenta de que es idéntica a la revisión antigua, tan sólo se da cuenta de que se ha modificado). Si mperez quiere estar aún más seguro, puede hacer un diff para comprobarlo:

```
paste$ cvs -Q diff -c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.4
diff -c -r1.4 hello.c
*** hello.c     1999/04/20 04:14:37     1.4
--- hello.c     1999/04/20 06:02:25
***************
*** 4,9 ****
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("ENTRE HOLA Y ADIÓS.\n");
    printf ("Adiós, mundo!\n");
  }
--- 4,9 --
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("entre hola y adiós\n");
    printf ("Adiós, mundo!\n");
  }
paste$
```

Sí, esto es exactamente lo que él quería: una verdadera vuelta atrás. De hecho, es lo contrario del diff que obtuvo previamente. Satisfecho, lo envía:

```
paste$ cvs ci -m "devuelto al código 1.3"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <-  hello.c
new revision: 1.5; previous revision: 1.4
done
paste$
```

El método rápido de deshacer cosas

La forma rápida y vistosa de deshacer es usar la opción -j (de "juntar") al comando de actualización. Esta opción es similar a -r en el sentido de que utiliza un número de revisión, y de que puede usar hasta dos -j a la vez. CVS calcula la diferencia entre las dos revisiones nombradas y aplica esta diferencia como un parche al fichero en cuestión (con lo cual, será de vital importancia el orden en el que indique estas revisiones al comando).

Así pues, asumiendo que la copia de mperez está al día, puede hacer simplemente lo siguiente:

```
paste$ cvs update -j 1.4 -j 1.3 hello.c
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.4
retrieving revision 1.3
Merging differences between 1.4 and 1.3 into hello.c
paste$ cvs update
cvs update: Updating .
M hello.c
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
paste$ cvs ci -m "devuelto al código 1.3" hello.c
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <--  hello.c
new revision: 1.5; previous revision: 1.4
done
paste$
```

Cuando sólo es necesario deshacer los cambios habidos en un único fichero, no hay realmente mucha diferencia entre el método lento y el rápido. Más adelante, veremos como el método rápido es mucho mejor para deshacer múltiples ficheros al mismo tiempo. De momento, utilice simplemente el método que le resulte más cómodo.

Deshacer no es un sustituto para la comunicación

Con toda probabilidad, lo que mperez hizo en nuestro ejemplo fue bastante desconsiderado por su parte. Al trabajar en un proyecto real con otras personas, si se considera que alguien ha hecho un cambio poco deseable, lo primero que debería hacerse es discutir el tema con esa persona. Quizás haya una buena razón detrás del cambio, o puede que esa persona no lo haya meditado lo suficiente antes de incorporarlo al repositorio. Sea como fuere, no hay motivo alguno para precipitarse a deshacerlo. CVS guarda un registro completo de todo lo que va sucediendo, así que siempre puede deshacer los cambios hasta una revisión previa después de consultar con la persona que hizo esos cambios.

Si es Ud. el responsable máximo de un proyecto y tiene una fecha límite para entregarlo, o si considera que tiene todo el derecho -y la necesidad- de deshacer inmediatamente el cambio sin más dilación, entonces adelante, pero lo siguiente que debería hacer es contactar con el autor de los cambios que ha deshecho, explicándole sus motivos y qué hay que hacer para volver a enviar el cambio.

Otros comandos útiles de CVS

Llegados a este punto, ya debería sentirse relativamente cómodo con el uso básico de CVS. Dejaré por tanto mi tono narrativo y docente para presentar algunos comandos más que resultan útiles, esta vez de forma más breve:

Añadir ficheros

Añadir un fichero es un proceso de dos pasos: primero debe ejecutar el comando add ("añadir") sobre él, y después enviarlo. El fichero no aparecerá en el repositorio hasta que se haya realizado el envío:

```
floss$ cvs add nuevofichero.c
cvs add: scheduling file 'nuevofichero.c' for addition
cvs add: use 'cvs commit' to add this file permanently
floss$ cvs ci -m "añadido nuevofichero.c" nuevofichero.c
RCS file: /usr/local/cvs/miproyecto/nuevofichero.c,v
done
Checking in nuevofichero.c;
/usr/local/cvs/miproyecto/nuevofichero.c,v  <-  nuevofichero.c
initial revision: 1.1
done
floss$
```

Añadir directorios

A diferencia del añadido de ficheros, el añadido de directorios es un proceso de un sólo paso; no es necesario emplear "commit" a continuación:

```
floss$ mkdir c-subdir
floss$ cvs add c-subdir
Directory /usr/local/cvs/miproyecto/c-subdir added to the repository
floss$
```

Si examina lo que hay dentro del nuevo directorio en la copia de trabajo, verá que el comando "add" ha añadido automáticamente un subdirectorio "CVS":

```
floss$ ls c-subdir
CVS/
floss$ ls c-subdir/CVS
Entries     Repository  Root
floss$
```

Ahora puede añadir ficheros (o nuevos directorios) al directorio que ha creado, como con cualquier otro directorio de su copia de trabajo.

CVS y ficheros binarios

Hasta ahora he decidido omitir el secretillo que esconde CVS, que es el hecho de que no se le dan muy bien los ficheros binarios (bueno, hay algún que otro secretillo más, pero éste es desde luego el más vergonzante de todos). No es que CVS no soporte el uso de binarios; en realidad lo hace, pero no sin algunos dolores de cabeza.

Todos los ficheros en los que hemos estado trabajando hasta ahora han sido simples ficheros de texto. CVS tiene algunos trucos especiales para los ficheros de texto: por ejemplo, cuando está trabajando entre un repositorio de Unix y una copia de trabajo instalada en un sistema Windows o Macintosh, convierte los finales de línea de la forma apropiada para cada plataforma. Por ejemplo, la convención en Unix es usar simplemente un salto de línea (LF), mientras que Windows espera una secuencia de retorno de carro y salto de línea (CR+LF)

al final de cada línea. Así, los ficheros en una copia de trabajo bajo Windows tendrán terminaciones CR+LF, mientras que una copia de trabajo del mismo proyecto instalada en una máquina Unix tendrá terminaciones LF (el repositorio en sí siempre utiliza el formato LF).

Otro truco es que CVS detecta en los ficheros de texto la presencia de cadenas especiales, conocidas como cadenas de texto de palabras clave RCS, y las sustituye con la información de la revisión y otras cosas útiles. Por ejemplo, si su fichero contiene esta cadena:

```
$Revision$
```

CVS la expandirá en cada envío para poner en su lugar el número de revisión. Por ejemplo, esta cadena podría convertirse en

```
$Revision: 1.3 $
```

CVS mantiene esta cadena al día en el fichero a medida que éste va evolucionando. Estas palabras clave se documentan en ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩ y ⟨undefined⟩ [Herramientas de terceros], page ⟨undefined⟩.

Esta expansión de cadenas de texto es algo muy útil en los ficheros de texto, ya que permite ver los números de revisión y otra información sobre un fichero mientras lo está editando. Ahora bien, qué pasa si el fichero es una imagen JPG? o un programa ejecutable compilado? En estos casos, CVS podría dañar seriamente los ficheros si se pusiera a expandir cada palabra clave que fuese encontrando. Es más, en un fichero binario estas cadenas podrían aparecer por pura coincidencia.

Por lo tanto, cuando añada un fichero binario, debe decirle a CVS que desactive tanto la expansión de palabras clave como la conversión de finales de línea. Para ello, utilice -kb:

```
floss$ cvs add -kb fichero
floss$ cvs ci -m "añadido esto y lo otro" fichero
  (etcétera)
```

Por otra parte, en ciertas ocasiones (como es el caso de ficheros de texto donde posiblemente aparezca alguna referencia a este tipo de palabras clave), posiblemente desee desactivar solamente la expansión de palabras clave. Esto se hace con -ko:

```
floss$ cvs add -ko fichero
floss$ cvs ci -m "añadido esto y lo otro" fichero
  (etcétera)
```

(De hecho, este mismo capítulo que está leyendo es un ejemplo de este tipo de casos, merced al ejemplo de `$Revision$` mostrado aquí.)

Tenga en cuenta que no tiene sentido utilizar `cvs diff` sobre dos revisiones de un fichero binario. Diff utiliza un algoritmo basado en texto que sólo puede discernir si dos ficheros binarios son distintos, pero no la forma en que difieren. Futuras versiones de CVS podrían llegar a ofrecer alguna manera de presentar diferencias entre ficheros binarios.

Eliminar ficheros

Eliminar un fichero es similar a añadir uno, con la salvedad de que debe dar un paso adicional: tendrá que eliminar antes el fichero de su copia de trabajo.

```
floss$ rm nuevofichero.c
floss$ cvs remove nuevofichero.c
cvs remove: scheduling 'nuevofichero.c' for removal
```

```
cvs remove: use 'cvs commit' to remove this file permanently
floss$ cvs ci -m "eliminado nuevofichero.c" nuevofichero.c
Removing nuevofichero.c;
/usr/local/cvs/miproyecto/nuevofichero.c,v  <-  nuevofichero.c
new revision: delete; previous revision: 1.1
done
floss$
```

Fíjese cómo en el segundo y tercer comandos nos referimos a nuevofichero.c de forma explícita a pesar de que ya no existe en nuestra copia de trabajo. Por supuesto, en el envío, no es estrictamente necesario que nombre el fichero, siempre que no le importe que el envío afecte también a cualquier otra modificación que haya podido tener lugar en la copia de trabajo.

Eliminar directorios

Como he dicho anteriormente, en realidad CVS no mantiene los directorios bajo control de versión, sino que, a modo de medida de ahorro, presenta ciertos comportamientos extraños que en la mayoría de los casos hacen simplemente "lo que conviene". Uno de estos comportamientos extraños es que los directorios vacíos pueden tratarse de una forma especial. Si desea eliminar un directorio de un proyecto, primero borre todos los ficheros que contenga

```
floss$ cd dir
floss$ rm fichero1 fichero2 fichero3
floss$ cvs remove fichero1 fichero2 fichero3
  (salida de información omitida)
floss$ cvs ci -m "borrados todos los ficheros" fichero1 fichero2 fichero3
  (salida de información omitida)
```

y después ejecute "update" en el directorio anterior con la opción -P:

```
floss$ cd ..
floss$ cvs update -P
  (salida de información omitida)
```

La opción -P le dice al comando "update" que "limpie" cualquier directorio vacío, esto es, que los elimine de la copia de trabajo. Una vez hecho esto, puede decirse que el directorio ha sido borrado; todos sus ficheros han desaparecido, y el directorio en sí es historia (al menos por lo que respecta a nuestra copia de trabajo, claro, pero sigue habiendo un directorio vacío en el repositorio).

Una interesante contrapartida de este comportamiento es que, cuando se hace una actualización normal, CVS no trae automáticamente a la copia de trabajo los nuevos directorios que se hayan creado en el repositorio. Hay un par de justificaciones variopintas para esto, ninguna de las cuales vale la pena mencionar aquí. La explicación breve es que de vez en cuando debería ejecutar "update" con la opción -d, ordenándole que traiga cualquier nuevo directorio que haya sido creado en el repositorio.

Renombrar ficheros y directorios

Renombrar un fichero es equivalente a crearlo con un nuevo nombre y eliminarlo con el antiguo. En Unix, los comandos son:

```
floss$ cp nombre_antiguo nuevo_nombre
floss$ rm nombre_antiguo
```

Aquí está el equivalente en CVS:

```
floss$ mv nombre_antiguo nuevo_nombre
floss$ cvs remove nombre_antiguo
  (salida omitida)
floss$ cvs add nuevo_nombre
  (salida omitida)
floss$ cvs ci -m "renombrado nombre_antiguo como nuevo_nombre" nombre_antiguo nuevo_no
  (salida omitida)
floss$
```

Para ficheros, ésto es todo lo que hay que hacer. El renombramiento de directorios tampoco varía mucho: deberá crear el nuevo directorio, añadirlo con cvs add, mover todos los ficheros del directorio antiguo al nuevo, eliminarlos con cvs remove del directorio antiguo, añadirlos al nuevo con cvs add, hacer un envío con cvs commit para actualizar el repositorio, y finalmente actualizarse con cvs update -P para que desaparezca de nuestra copia de trabajo cualquier directorio vacío. O sea,

```
floss$ mkdir nuevo_dir
floss$ cvs add nuevo_dir
floss$ mv dir_antiguo/* nuevo_dir
mv: nuevo_dir/CVS: cannot overwrite directory
floss$ cd dir_antiguo
floss$ cvs rm foo.c bar.txt
floss$ cd ../nuevo_dir
floss$ cvs add foo.c bar.txt
floss$ cd ..
floss$ cvs commit -m "movidos foo.c y bar.txt de dir_antiguo a nuevo_dir"
floss$ cvs update -P
```

Fíjese en el mensaje de aviso recibido después de ejecutar el tercer comando. Este aviso le comunica que no se puede copiar el subdirectorio CVS/ del antiguo directorio al nuevo porque ya hay allí un directorio con este nombre. Esto es correcto, porque le interesa que el directorio antiguo mantenga de todos modos su subdirectorio CVS/.

Obviamente, mover directorios de un lado para otro puede ser un poco engorroso. La mejor política es tratar de disponer una buena distribución en el momento de importar el proyecto, de forma que no tenga que estar moviendo luego directorios para aquí y para allá. Más adelante, aprenderá un método más drástico para mover directorios, esta vez haciendo el cambio directamente en el repositorio. Sin embargo, es mejor reservar este método para situaciones de emergencia; siempre que sea posible, es mejor gestionar todo con operaciones de CVS que afecten principalmente a las copias de trabajo.

Evitar la fatiga de las opciones

La mayor parte de la gente se cansa rápidamente de estar escribiendo continuamente las mismas opciones con cada comando. Si sabe que siempre va a querer pasar la opción global -Q o que siempre va a usar -c con diff, por qué tendría que estar escribiéndolas una y otra vez?

Por suerte, dispone de ayuda para esto. CVS busca un fichero .cvsrc en su directorio personal. En este fichero puede especificar las opciones que deban aplicarse por omisión en cada invocación a CVS. Éste es un fichero .cvsrc de ejemplo:

```
diff -c
update -P
cvs -q
```

Si la primera palabra de la línea corresponde a un comando de CVS en su forma *no* abreviada, las opciones indicadas ahí serán las que se utilicen siempre con ese comando. Para indicar opciones globales a emplear por omisión, deberá usar "cvs".

Empleando nuestro ejemplo de arriba, cada vez que el usuario ejecute cvs diff, la opción -c se incluirá automáticamente.

Obtener instantáneas (fechas y marcas)

Volvamos al ejemplo del programa que está en un estado inoperativo en el momento en el que recibimos un informe de fallo de un usuario. El desarrollador necesita de repente acceder al proyecto entero en el estado en el que se encontraba al hacer la última distribución pública, aunque muchos ficheros posiblemente habrán cambiado desde entonces, y el número de revisión de cada fichero seguramente será distinto en este momento. Llevaría demasiado tiempo revisar los informes de cambios, adivinar cuál era el número de revisión de cada fichero en el momento de la distribución, y después ejecutar una actualización (especificando el número de revisión con -r) para cada uno de ellos. En proyectos de mediano o gran tamaño (de decenas a cientos de ficheros), sería casi imposible acometer tamaña empresa.

Por todo lo cual, CVS proporciona una forma de obtener de una sola vez revisiones antiguas de los ficheros que forman un proyecto. De hecho, ofrece dos mecanismos: por fecha -que selecciona las revisiones basándose en la fecha en que fueron enviadas al repositorio-, y por marca, que obtiene una "instantánea" del proyecto marcada previamente.

Cuál de los métodos deberá utilizar es algo que depende de la situación concreta. El método basado basado en fechas funciona pasando la opción -D al comando "update", que es similar a -r pero en este caso tomando como argumento fechas en lugar de números de revisión:

```
floss$ cvs -q update -D "1999-04-19"
U hello.c
U a-subdir/subsubdir/fish.c
U b-subdir/random.c
floss$
```

Con la opción -D, "update" recupera la revisión más reciente de cada fichero existente en la fecha dada, devolviendo si es necesario los ficheros de la copia de trabajo al estado en el que encontes se encontraban.

Cuando se da una fecha, puede -y generalmente, debería hacerlo- incluir la hora exacta. Por ejemplo, el comando anterior terminó tomando del repositorio la revisión 1.1 de todo (sólo tres ficheros cambiaron, porque los demás están aún en la revisión 1.1). Éste es el estado de hello.c para demostrarlo:

```
floss$ cvs -Q status hello.c
===================================================================
File: hello.c                 Status: Up-to-date
```

```
   Working revision:          1.1.1.1 Sat Apr 24 22:45:03 1999
   Repository revision:       1.1.1.1 /usr/local/cvs/miproyecto/hello.c,v
   Sticky Date:               99.04.19.05.00.00
floss$
```

Pero un vistazo atrás a los informes de cambios presentados anteriormente en este capítulo indica que la revisión 1.2 de hello.c se envió sin duda alguna al repositorio el 19 de Abril de 1999. Así que, por qué hemos recibido ahora la revisión 1.1 en lugar de la 1.2?

El problema está en que la fecha "1999-04-19" fue interpretada como "la medianoche en la que comenzó el día 1999-04-19", esto es, el primer instante de ese día. Esto seguramente no es lo que quiere. El envío de la revisión 1.2 tuvo lugar en un momento posterior de ese día. Indicando la fecha con mayor exactitud, podemos obtener la revisión 1.2:

```
floss$ cvs -q update -D "1999-04-19 23:59:59"
U hello.c
U a-subdir/subsubdir/fish.c
U b-subdir/random.c
floss$ cvs status hello.c
===================================================================
File: hello.c                  Status: Locally Modified
   Working revision:  1.2      Sat Apr 24 22:45:22 1999
   Repository revision:       1.2     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                (none)
   Sticky Date:               99.04.20.04.59.59
   Sticky Options:    (none)
floss$
```

Casi hemos terminado. Si observa atentamente la fecha y hora de la línea "Sticky Date", parece indicar las 4:59:59 AM, no las 11:59 como se indicó en el comando (más adelante veremos lo que significa "sticky").

Como habrá adivinado, la discrepancia se debe a la diferencia entre la hora local y la hora Universal Coordinada (también conocida como la hora del meridiano de Greenwich). El repositorio guarda siempre las fechas en Tiempo Universal, pero en su lado cliente CVS suele asumir la hora local del sistema. En el caso de -D, esto es un poco desafortunado, porque posiblemente le interese más comparar respecto a la hora recogida en el repositorio que preocuparse por lo que el sistema local opina de qué hora es. Puede evitar este problema especificando la zona GMT en el comando:

```
floss$ cvs -q update -D "1999-04-19 23:59:59 GMT"
U hello.c
floss$ cvs -q status hello.c
===================================================================
File: hello.c                  Status: Up-to-date
   Working revision:  1.2      Sun Apr 25 22:38:53 1999
   Repository revision:       1.2     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                (none)
   Sticky Date:               99.04.19.23.59.59
   Sticky Options:    (none)
floss$
```

Ajá! Esto ha hecho que la copia de trabajo quede registrada como uno de los últimos envíos realizados el 19 de Abril (a menos que hubiera algún envío en el último segundo del día, que no es el caso).

Qué sucede ahora si ejecuta "update"?

```
floss$ cvs update
cvs update: Updating .
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
floss$
```

No pasa nada. Pero usted sabe que hay versiones más recientes de por lo menos tres ficheros, luego, por qué no se incorporan estos ficheros en su copia de trabajo?

Aquí es donde "sticky" entra en juego. Actualizar ("desactualizar"?) con la opción -D hace que la copia de trabajo quede fijada de forma permanente a esa fecha o a una anterior: en terminología de CVS, la copia de trabajo tiene "pegada" una fecha. Una vez que una copia de trabajo ha adquirido una propiedad pegadiza, se mantiene así hasta que se le diga lo contrario. Así pues, posteriores actualizaciones no permitirán obtener automáticamente la revisión más reciente, sino que estarán restringidas a la fecha pegada. El si hay algo pegado o no a un fichero es algo que puede averiguarse ejecutando cvs status, o bien examinando directamente el fichero CVS/Entries:

```
floss$ cvs -q update -D "1999-04-19 23:59:59 GMT"
U hello.c
floss$ cat CVS/Entries
D/a-subdir////
D/b-subdir////
D/c-subdir////
/README.txt/1.1.1.1/Sun Apr 18 18:18:22 1999//D99.04.19.23.59.59
/hello.c/1.2/Sun Apr 25 23:07:29 1999//D99.04.19.23.59.59
floss$
```

Si ahora usted modificase su copia de hello.c e intentase después enviarla al repositorio,

```
floss$ cvs update
M hello.c
floss$ cvs ci -m "intentando cambiar el pasado"
cvs commit: cannot commit with sticky date for file 'hello.c'
cvs [commit aborted]: correct above errors first!
floss$
```

CVS no permitiría la enviar ese fichero al repositorio, puesto que sería como permitirle volver atrás y cambiar el pasado. Lo de CVS es gestionar historiales, así que bajo ningún concepto le permitirá realizar semejante operación.

Esto no significa, sin embargo, que CVS no sea consciente de todas las revisiones que se hayan enviado desde aquella fecha. Usted sigue pudiendo comparar la fecha pegada a la copia de trabajo con otras revisiones, incluso las futuras:

```
floss$ cvs -q diff -c -r 1.5 hello.c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
```

```
retrieving revision 1.5
diff -c -r1.5 hello.c
*** hello.c   1999/04/24 22:09:27     1.5
--- hello.c   1999/04/25 00:08:44
***************
*** 3,9 ****
  void
  main ()
  {
    printf ("Hola, mundo!\n");
-   printf ("entre hola y adiós\n");
    printf ("Adiós, mundo!\n");
  }
--- 3,9 --
  void
  main ()
  {
+   /* esta línea fue añadida a una copia de trabajo "desactualizada" */
    printf ("Hola, mundo!\n");
    printf ("Adiós, mundo!\n");
  }
```

El diff revela que, a 19 de abril de 1999, la línea "entre hola y adiós" no había sido añadida todavía. También muestra la modificación que hicimos a la copia de trabajo (añadir el comentario que aparece arriba).

Puede eliminar la fecha pegadiza (o cualquier otra propiedad pegadiza) haciendo una actualización con la opción -A ("-A" significa "reiniciar", no me pregunte por qué), lo que pone la copia de trabajo de nuevo en sincronía con las revisiones más recientes:

```
floss$ cvs -q update -A
U hello.c
floss$ cvs status hello.c
===================================================================
File: hello.c                 Status: Up-to-date
   Working revision:  1.5     Sun Apr 25 22:50:27 1999
   Repository revision:       1.5     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                (none)
   Sticky Date:               (none)
   Sticky Options:    (none)
floss$
```

Formatos de fecha permitidos

CVS admite una gran variedad de formatos cuando se trata de indicar fechas. Nunca tendrá problemas si decide utilizar el formato ISO 8601 (esto es, en la Organización Internacional de Estándares, el estándar número 8601, consulte también http://www.saqqara.demon.co.uk/datefmt.htm), que es el formato empleado en los ejemplos precedentes. Puede también utilizar el formato de fecha empleado en el correo electrónico vía Internet tal como se describe en los RFC 822 y 1123 (véase

http://www.rfc-editor.org/rfc/). Por último, puede utilizar ciertas construcciones anglosajonas no ambiguas para especificar una fecha relativa a la fecha actual.

Lo más probable es que nunca necesite utilizar todos los formatos disponibles, pero aquí van algunos ejemplos para darle una idea de lo que admite CVS:

```
floss$ cvs update -D "19 Apr 1999"
floss$ cvs update -D "19 Apr 1999 20:05"
floss$ cvs update -D "19/04/1999"
floss$ cvs update -D "3 days ago"
floss$ cvs update -D "5 years ago"
floss$ cvs update -D "19 Apr 1999 23:59:59 GMT"
floss$ cvs update -D "19 Apr"
```

Las comillas que delimitan las fechas están ahí para asegurar que el shell de Unix trata la fecha como un único argumento aunque incluya espacios en blanco. Las comillas no causarán ningún problema aunque la fecha no contenga espacios, así que probablemente sea mejor usarlas siempre.

Marcar un momento en el tiempo (marcas)

Pedir ficheros según una fecha concreta es útil cuando el mero paso del tiempo es su principal preocupación. Pero normalmente lo que realmente querrá será obtener el proyecto tal y como estaba al producirse un determinado evento concreto: quizás el día de hacer una distribución pública, un punto en el que el proyecto se encontraba en su momento más estable, o la vez que se añadió o eliminó cierta característica importante.

Intentar recordar la fecha en la que tuvo lugar un evento en concreto o deducir la fecha a partir de los informes de cambios sería un proceso muy tedioso. Presumiblemente, el evento, en caso de que fuese realmente importante, se marcó como tal en el historial formal de revisiones. El método que ofrece CVS para crear este tipo de marcas se conoce como *marcado*.

Las marcas ("tags", N. del T.) se diferencian de los envíos corrientes en que no registran ningún cambio particular en el texto de los ficheros, sino más bien un cambio en la forma en que los desarrolladores contemplan los ficheros. Lo que hace una marca es poner una etiqueta al conjunto de revisiones representadas por la copia de trabajo de un desarrollador (lo normal es que la copia de trabajo está completamente al día, así que el nombre de la marca se pone a las revisiones "más recientes y brillantes" presentes en el repositorio).

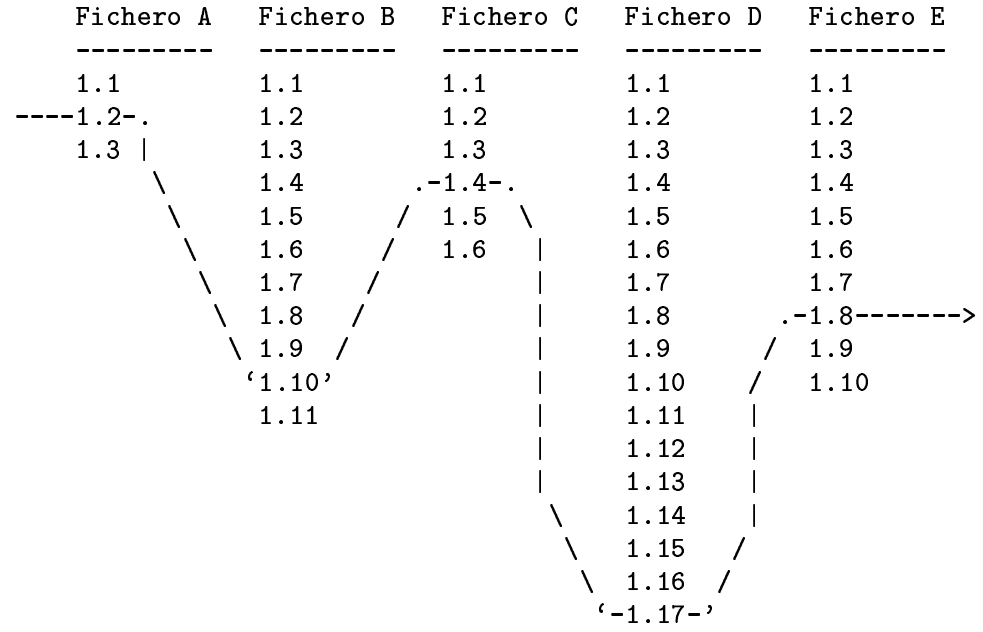
Poner una marca es tan sencillo como ésto:

```
floss$ cvs -q tag Release-1999_05_01
T README.txt
T hello.c
T a-subdir/loquesea.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
floss$
```

Este comando asocia el nombre simbólico "Release-1999_05_01" a la instantánea representada por esta copia de trabajo. Definida formalmente, una "instantánea" es un conjunto de ficheros del proyecto y los números de revisión asociados con ellos. Estos números de revisión no tienen por qué ser los mismos entre varios ficheros, y de hecho no suelen serlo. Por

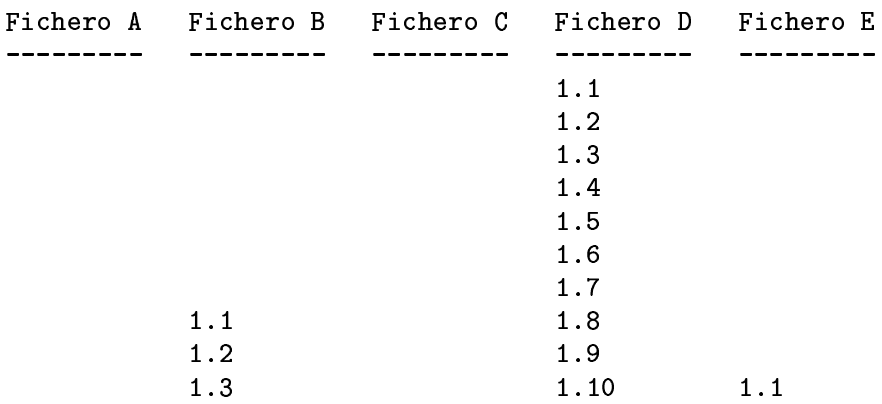
ejemplo, suponiendo que la marca se hubiera puesto en el mismo directorio "miproyecto" que hemos estado utilizando en este capítulo y que la copia de trabajo estuviese completamente al día, el nombre simbólico "Release-1999_05_01" se asociaría a la revisión 1.5 de hello.c, a la revisión 1.2 de fish.c, a la revisión 1.2 de random.c, y a la revisión 1.1 de todo lo demás.

Quizás le ayude el pensar en una marca como en una senda o cadena que une varias revisiones de ficheros del proyecto. En la Figura 2.1, una cadena imaginaria pasa por el número de revisión marcado en cada fichero del proyecto.



[Figura 2.1: La relación que puede haber entre una marca y determinadas revisiones de los ficheros.]

Ahora, si estira la cadena y la observa de forma longitudinal, podrá ver un momento específico en el historial del proyecto; concretamente, el momento en el que se fijó la marca (Figura 2.2).



```
                 1.4                   1.11       1.2
                 1.5                   1.12       1.3
                 1.6                   1.13       1.4
                 1.7        1.1        1.14       1.5
                 1.8        1.2        1.15       1.6
     1.1         1.9        1.3        1.16       1.7
 ----1.2---------1.10--------1.4---------1.17--------1.8------->
     1.3         1.11       1.5        1.17       1.9
                            1.6        1.17       1.10
```

[Figura 2.2: La misma marca vista longitudinalmente respecto al historial de revisiones.]

A medida que sigue editando ficheros y enviando cambios, la marca *no* se moverá de su sitio a pesar de que los números de revisión se vayan incrementando. Permanece fija, "pegada" al número de revisión de cada fichero al que fue asociada.

Dada su importancia como elementos descriptivos, es una pena que los informes de cambios no puedan ser incluidos con las marcas, y que las marcas en sí no puedan ser párrafos de texto reales. En el ejemplo anterior, la marca indica de forma bastante obvia que el proyecto se encontraba en un estado "publicable" en una fecha determinada. Sin embargo, a veces querrá hacer instantáneas de un estado más complejo, lo que puede dar lugar a nombres de marcas un tanto extraños, como:

```
floss$ cvs tag testing-release-3_pre-19990525-public-release
```

Por regla general, debería intentar mantener las marcas lo más simples posible al tiempo que incluye toda la información necesaria sobre el evento que está intentando registrar. En caso de duda, es mejor pecar de demasiado descriptivos - lo agradecerá más tarde cuando sea capaz de determinar gracias a una marca extremadamente descriptiva exactamente qué circunstancia se registró con ella.

Posiblemente se haya dado cuenta de que no se han usado puntos ni espacios en el nombre de las marcas. CVS es bastante estricto en lo que atañe a nombres válidos de marcas: las reglas son que deben comenzar con una letra y tener letras, dígitos, guiones ("-"), y signos de subrayado ("_"). No se pueden utilizar espacios, puntos, signos de dos puntos, comas ni ningún otro símbolo.

Para obtener una instantánea haciendo referencia a una marca, deberá usar el nombre de esta marca como si fuera un número de revisión. Hay dos formas de obtener instantáneas: puede obtener una copia de trabajo nueva con una determinada marca, o bien puede cambiar una copia de trabajo ya existente por una marca. Ambos métodos tienen como resultado una copia de trabajo en la que las revisiones de los ficheros son los asociados a la marca dada.

La mayor parte de las veces, lo que querrá hacer es echar un vistazo al proyecto tal como éste se encontraba en el momento de hacer la instantánea. No necesariamente querrá hacer esto en su copia de trabajo principal, donde posiblemente tenga cambios aún sin enviar y otras cosas importantes que preferirá salvaguardar, así que vamos a suponer que simplemente desea obtener una copia de trabajo por separado, con la marca. Así es cómo

se hace (asegúrese de invocar este comando desde cualquier otro lugar que no sea su copia de trabajo actual o su directorio padre!):

```
floss$ cvs checkout -r Release-1999_05_01 miproyecto
cvs checkout: Updating miproyecto
U miproyecto/README.txt
U miproyecto/hello.c
cvs checkout: Updating miproyecto/a-subdir
U miproyecto/a-subdir/loquesea.c
cvs checkout: Updating miproyecto/a-subdir/subsubdir
U miproyecto/a-subdir/subsubdir/fish.c
cvs checkout: Updating miproyecto/b-subdir
U miproyecto/b-subdir/random.c
cvs checkout: Updating miproyecto/c-subdir
```

Anteriormente hemos visto cómo se utilizaba la opción -r con el comando "update", donde precedía a un número de revisión. En muchos aspectos, una marca es como un número de revisión, porque, para cada fichero, cada marca se corresponde exactamente con *un* número de revisión concreto (es ilegal, y generalmente imposible, tener dos marcas con el mismo nombre en el mismo proyecto). De hecho, en cualquier lugar donde pueda utilizar un número de revisión como argumento de un comando de CVS, podrá también utilizar el nombre de una marca, siempre y cuando esa marca se haya creado antes. Si quiere ver las diferencias entre el estado actual de un fichero y el estado en el que se encontraba en el momento de realizar la última distribución pública, puede hacer esto:

```
floss$ cvs diff -c -r Release-1999_05_01 hello.c
```

Y si quiere volver atrás temporalmente a aquella revisión, puede hacer esto:

```
floss$ cvs update -r Release-1999_05_01 hello.c
```

La posibilidad de intercambiar nombres de marcas y números de revisión explica algunas de las estrictas reglas que definen qué nombre de marca es válido y cuál no. Imagine por un momento que los puntos fueran legales en los nombres de las marcas; podría tener una marca llamada "1.3" asociada a un número real de revisión como puede ser el "1.47". Si ahora emplease el comando

```
floss$ cvs update -r 1.3 hello.c
```

cómo podría CVS saber si se refiere a la marca llamada "1.3", o a la mucho más antigua revisión 1.3 de hello.c? Es por esto por lo que existen este tipo de restricciones en los nombres de las marcas, de forma que siempre puedan distinguirse fácilmente de los números de revisión. Un número de revisión tiene un punto; el nombre de una marca no. (También hay motivos para las demás restricciones, generalmente para permitir que los nombres de las marcas sean fáciles de interpretar para CVS.)

Como posiblemente habrá ya adivinado, el segundo método para obtener una instantánea -esto es, pasar un directorio de trabajo ya existente a las revisiones marcadas- también se realiza por medio de una actualización:

```
floss$ cvs update -r Release-1999_05_01
cvs update: Updating .
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
cvs update: Updating c-subdir
```

```
floss$
```

Este comando es similar al que utilizamos para devolver hello.c al estado en el que se encontraba en `Release-1999_05_01`, con la salvedad de que se omite el nombre del fichero dado que queremos volver atrás el proyecto al completo (si lo desea, puede hacer volver atrás solamente un sub-árbol del proyecto, invocando este comando mientras se encuentra en ese subdirectorio en lugar de hacerlo desde el nivel más alto, aunque muy pocas veces querrá hacer esto).

Observe que, a juzgar por los resultados del comando, no parece que haya cambiado ningún fichero. La copia de trabajo estaba completamente al día cuando pusimos la marca, y no se había enviado ningún cambio desde entonces.

Sin embargo, esto no significa que no haya habido ningún cambio en absoluto. La copia de trabajo sabe ahora que responde a una revisión marcada. Cuando hagamos un cambio e intentemos enviarlo al repositorio (supongamos que hemos modificado hello.c):

```
floss$ cvs -q update
M hello.c
floss$ cvs -q ci -m "intentando enviar desde una copia de trabajo marcada"
cvs commit: sticky tag 'Release-1999_05_01' for file 'hello.c' is not a branch
cvs [commit aborted]: correct above errors first!
floss$
```

... CVS impide que el envío llegue a producirse (no se preocupe de momento por lo que significa el mensaje de error - más adelante y en este mismo capítulo trataremos el tema de las "branches", o ramas para entendernos). Poco importa si la copia de trabajo llegó a tener una marca a través de la obtención de una copia de trabajo ("checkout") o una simple actualización; la cuestión es que, una vez que está marcada, CVS considera la copia de trabajo como una instantánea estática de un momento particular de la historia, y CVS no le permitirá cambiar la historia, o cuando menos no se lo pondrá fácil. Si ejecuta "cvs status" o examina los ficheros CVS/Entries, podrá ver que hay una marca pegada a cada fichero. Por ejemplo, aquí tiene el fichero Entries del directorio raíz:

```
floss$ cat CVS/Entries
D/a-subdir////
D/b-subdir////
D/c-subdir////
/README.txt/1.1.1.1/Sun Apr 18 18:18:22 1999//TRelease-1999_05_01
/hello.c/1.5/Tue Apr 20 07:24:10 1999//TRelease-1999_05_01
floss$
```

Las marcas, como otras propiedades de carácter "pegadizo", pueden eliminarse con la opción -A durante una actualización:

```
floss$ cvs -q update -A
M hello.c
floss$
```

Sin embargo, la modificación de hello.c no ha desaparecido; CVS todavía sabe que el fichero ha cambiado respecto a su homónimo en el repositorio:

```
floss$ cvs -q diff -c hello.c
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
```

```
retrieving revision 1.5
diff -c -r1.5 hello.c
*** hello.c   1999/04/20 06:12:56     1.5
--- hello.c   1999/05/04 20:09:17
***************
*** 6,9 ****
--- 6,10 --
    printf ("Hola, mundo!\n");
    printf ("entre hola y adiós\n");
    printf ("Adiós, mundo!\n");
+   /* un comentario en la última línea */
  }
floss$
```

Ahora que ha puesto las cosas en orden por medio de la actualización, CVS aceptará hacer un envío:

```
floss$ cvs ci -m "añadido un comentario al final de la función main"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
cvs commit: Examining c-subdir
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <-  hello.c
new revision: 1.6; previous revision: 1.5
done
floss$
```

Por supuesto, la marca `Release-1999_05_01` sigue estando asociada a la revisión 1.5. Compare el estado del fichero de como estaba antes a como está después de eliminar la marca:

```
floss$ cvs -q status hello.c
===================================================================
File: hello.c                 Status: Up-to-date
   Working revision:  1.6     Tue May  4 20:09:17 1999
   Repository revision:       1.6     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                (none)
   Sticky Date:               (none)
   Sticky Options:            (none)
floss$ cvs -q update -r Release-1999_05_01
U hello.c
floss$ cvs -q status hello.c
===================================================================
File: hello.c                 Status: Up-to-date
   Working revision:  1.5     Tue May  4 20:21:12 1999
   Repository revision:       1.5     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                Release-1999_05_01 (revision: 1.5)
   Sticky Date:               (none)
   Sticky Options:            (none)
floss$
```

Y ahora que vengo de decirle que CVS no le permite cambiar la historia, le enseñaré precisamente cómo cambiarla.

Ramas

Hemos estado concibiendo CVS como un tipo de biblioteca inteligente desde la que establecer una coordinación; sin embargo, también puede pensarse en CVS como en una máquina del tiempo (gracias a Jim Blandy por la analogía). Por ahora, sólo hemos visto cómo se puede revisar el pasado con CVS, sin afectar a nada. Como todas las buenas máquinas del tiempo, CVS también le permite ir hacia atrás en el tiempo para cambiar el pasado. Y a dónde nos lleva esto? Los amantes de la ciencia-ficción conocen la respuesta: a un universo alternativo, que discurre de forma paralela al nuestro, pero que diverge del nuestro justo desde el punto en el que se alteró el pasado. Una rama de CVS divide el desarrollo de un proyecto en historias separadas, y paralelas. Los cambios efectuados en una de las ramas no afectan a las demás.

Conceptos básicos sobre ramas

Por qué son útiles las ramas?

Volvamos por un momento a la situación del desarrollador que, mientras está trabajando en una nueva versión del programa, recibe un informe de fallo relativo a una versión publicada anteriormente. Suponiendo que el desarrollador corrija el problema, aún tiene que encontrar una forma de enviar la corrección al cliente. No le servirá de nada limitarse a tomar una copia vieja del programa, parchearla a espaldas de CVS, y enviarla tal cual: no quedaría registro alguno de lo que ha hecho, CVS no sabría nada de esta corrección, y más adelante, si por un casual se descubriera un fallo en el propio parche, nadie tendría un punto desde el que comenzar a intentar reproducir el problema.

Es incluso peor intentar solucionar el fallo en la versión actual e inestable de las fuentes y entregar esto al cliente. Oh sí, el fallo del que se ha dado parte quizás quedase resuelto, pero el resto del código está a medio cocer y por lo general falto de un proceso de pruebas pertinente. Puede darse el caso de que funcione, pero es seguro que no está listo para llegar al gran público.

Dado que se supone que la última versión distribuida es estable (dejando aparte este fallo), la solución ideal es ir atrás y corregir el fallo en la antigua versión; esto es, crear un universo alternativo en el que la última versión pública incluye la correspondiente corrección.

Y aquí es donde entran en juego las ramas. El desarrollador planta una rama, que parte de la línea principal de desarrollo (el tronco), no en su revisiones más recientes, sino en el punto de la última distribución pública. Entonces el desarrollador solicita una copia de trabajo de esta rama, realiza todos los cambios necesarios para solventar el fallo, y los envía a esa rama, de forma que quede un registro de la corrección del fallo. Ahora puede hacer público un parche intermedio basado en esta rama, y enviarlo al cliente.

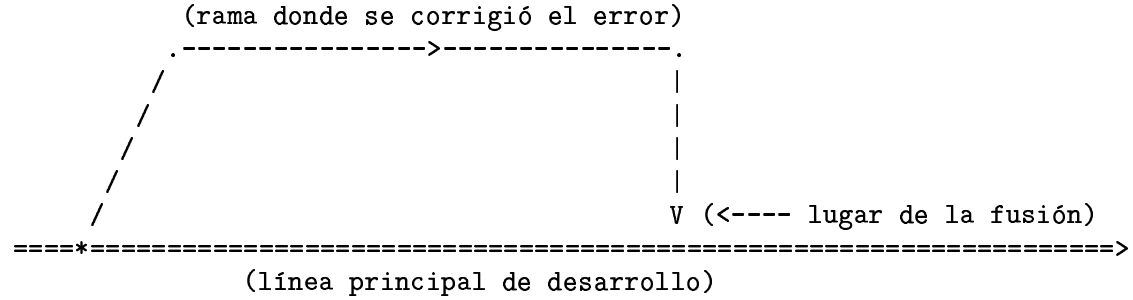
Su cambio no habrá afectado al código del tronco, puesto que tampoco le interesaría hacerlo sin antes averiguar si el tronco necesita o no que se aplique sobre él la misma corrección. En caso positivo, el desarrollador puede fusionar los cambios realizados sobre la rama con el código presente en el tronco. Durante el proceso de fusión, CVS calcula los cambios hechos en la rama desde el momento en el que ésta salió del tronco hasta el extremo

de la misma (su estado más reciente), y entonces aplica estas diferencias al proyecto, en el extremo final del tronco. La diferencia entre la raíz de la rama y su extremo final vendría a ser, por supuesto, la corrección que se ha realizado.

Otra buena forma de imaginar una fusión es como un caso especial del proceso de actualización; la diferencia estriba en que, durante una fusión, los cambios a incorporar se derivan de la comparación entre la raíz y el extremo de la rama, en lugar de comparar la copia de trabajo con el contenido del repositorio.

El proceso de actualización es en sí mismo similar a recibir parches directamente de los autores, y aplicarlos a mano; de hecho, para hacer una actualización, CVS calcula la diferencia (entendiendo como "diferencia" el resultado que devolvería el comando diff al comparar dos ficheros distintos) entre la copia de trabajo y el repositorio, para a continuación aplicar la diferencia a la copia de trabajo tal y como lo haría el programa "patch". Esto equivale a la forma en que un desarrollador obtiene cambios del mundo exterior, aplicando manualmente parches creados por otros.

Así pues, fusionar con el tronco la rama donde se ha hecho la corrección es exactamente lo mismo que aceptar un parche que ha hecho otra persona para corregir el fallo. El autor de ese parche habría generado el parche a partir de la última versión hecha pública, de la misma forma que los cambios en la rama lo son respecto a esa versión. Si esa parte del código en las fuentes actuales no ha variado mucho desde la última versión pública, la fusión tendrá éxito sin ningún problema. Ahora bien, si el código es en este momento lo suficientemente diferente, la fusión derivará en conflicto (en otras palabras, el parche será rechazado), y será necesario cierto trabajo extra. Normalmente esto se resuelve examinando la parte donde ha surgido el conflicto, haciendo manualmente los cambios necesarios, y enviando esos cambios al repositorio. La Figura 2.3 muestra gráficamente lo que sucede en una rama y en una fusión.



[Figura 2.3: Una rama que termina con una fusión. El tiempo transcurre de izquierda a derecha.]

Ahora vamos a ver los pasos necesarios para llevar a cabo el procedimiento descrito. Recuerde que no es realmente el tiempo lo que fluye de izquierda a derecha en el diagrama, sino más bien el historial de revisiones. La rama no se habrá hecho en el momento de la distribución, sino que es creada más tarde, aunque enraizada en las revisiones que formaban parte de la distribución.

En nuestro caso, supongamos que los ficheros del proyecto han pasado por muchas revisiones desde que fueron marcados como `Release-1999_05_01`, y que quizás se hayan añadido también nuevos ficheros. Al recibir el informe de fallos relativo a la antigua distribución, lo primero que querremos hacer será crear una rama que parta de la antigua distribución, que tuvimos el acierto de marcar conmo `Release-1999_05_01`.

Una forma de hacer esto sería obtener primero una copia de trabajo basada en dicha marca, y a continuación crear la rama volviendo a marcar con la opción -b (de "branch", o "rama" en inglés - N. del T.):

```
floss$ cd ..
floss$ ls
miproyecto/
floss$ cvs -q checkout -d miproyecto_antigua_dis -r Release-1999_05_01 miproyecto
U miproyecto_antigua_dis/README.txt
U miproyecto_antigua_dis/hello.c
U miproyecto_antigua_dis/a-subdir/loquesea.c
U miproyecto_antigua_dis/a-subdir/subsubdir/fish.c
U miproyecto_antigua_dis/b-subdir/random.c
floss$ ls
miproyecto/      miproyecto_antigua_dis/
floss$ cd miproyecto_antigua_dis
floss$ ls
CVS/      README.txt  a-subdir/   b-subdir/   hello.c
floss$ cvs -q tag -b Release-1999_05_01-bugfixes
T README.txt
T hello.c
T a-subdir/loquesea.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
floss$
```

Observe bien el último comando. Puede parecer un tanto arbitrario el utilizar "tag" para crear ramas, pero en realidad hay una razón para ello: el nombre de la marca servirá como una etiqueta con la cual podremos más tarde hacer alusión a esta rama. Las marcas en las ramas no tienen un aspecto diferente al de las utilizadas en otra parte, y de hecho están sujetas a las mismas limitaciones. A algunas personas les gusta incluir siempre la palabra "rama" en el nombre de la marca (por ejemplo `Release-1999_05_01-ramadecorrección`) para poder distinguir fácilmente entre marcas de rama y otros tipos de marcas. Tal vez le interese también a usted hacer lo mismo si alguna que otra vez se confunde y solicita la rama equivocada.

(Y ya que estamos, observe la opción -d miproyecto_antigua_dis que pasamos al comando "checkout" en la primer comando CVS. Esto le dice a checkout que instale la copia de trabajo en un directorio llamado miproyecto_antigua_dis, de forma que no confundamos estos ficheros con la versión actual de miproyecto. Tenga cuidado de no confundir este uso de la -d con la opción global homónima, o con la opción -d del comando "update".)

Por supuesto, la simple ejecución del comando "tag" no pone la copia de trabajo en consonancia con la rama. El hecho de marcar no afecta nunca a la copia de trabajo; tan sólo guarda información adicional en el repositorio para permitirle a usted recuperar en

un momento posterior las revisiones de esa copia de trabajo (como una parte estática del historial o como una rama, según el caso).

La recuperación puede hacerse de dos formas (a estas alturas, seguramente ya se esperaba oir esto). Puede solicitar una nueva copia de trabajo tomada de la rama:

```
floss$ pwd
/home/loquesea
floss$ cvs co -d miproyecto_rama -r Release-1999_05_01-bugfixes miproyecto
```

o pasar a ella una copia de trabajo ya existente:

```
floss$ pwd
/home/loquesea/miproyecto
floss$ cvs update -r Release-1999_05_01-bugfixes
```

El resultado final es el mismo (bueno, el nombre del directorio raíz de la nueva copia de trabajo puede ser distinto, pero respecto a los fines de CVS esto no importa). Si su copia de trabajo actual tiene cambios aún sin enviar, probablemente querrá utilizar "checkout" en lugar de "update" para acceder a la rama; de lo contrario, CVS intentará fusionar los cambios habidos en su copia de trabajo antes de colocarla en la rama. En este caso podría encontrarse con algún conflicto, y aún en caso de que no fuese así, seguiría sin tener una rama pura: esos ficheros no reflejarán realmente el estado del programa de acuerdo con la marca designada, puesto que algunos de ellos contendrán modificaciones hechas por usted.

Sea como fuere, vamos a suponer que de una forma o de otra usted obtiene una copia de trabajo operativa desde la rama deseada:

```
floss$ cvs -q status hello.c
===================================================================
File: hello.c                 Status: Up-to-date
   Working revision:  1.5     Tue Apr 20 06:12:56 1999
   Repository revision:       1.5     /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                Release-1999_05_01-bugfixes
(branch: 1.5.2)
   Sticky Date:               (none)
   Sticky Options:            (none)
floss$ cvs -q status b-subdir/random.c
===================================================================
File: random.c                Status: Up-to-date
   Working revision:  1.2     Mon Apr 19 06:35:27 1999
   Repository revision:       1.2 /usr/local/cvs/miproyecto/b-subdir/random.c,v
   Sticky Tag:                Release-1999_05_01-bugfixes (branch: 1.2.2)
   Sticky Date:               (none)
   Sticky Options:            (none)
floss$
```

(El contenido de las líneas `Sticky Tag` se explicará en breve.) Si modifica hello.c y random.c y envía los cambios al repositorio,

```
floss$ cvs -q update
M hello.c
M b-subdir/random.c
floss$ cvs ci -m "corregidos los viejos fallos de puntuación"
cvs commit: Examining .
```

```
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <-  hello.c
new revision: 1.5.2.1; previous revision: 1.5
done
Checking in b-subdir/random.c;
/usr/local/cvs/miproyecto/b-subdir/random.c,v  <-  random.c
new revision: 1.2.2.1; previous revision: 1.2
done
floss$
```

se dará cuenta de que ocurre algo curioso con los números de revisión:

```
floss$ cvs -q status hello.c b-subdir/random.c
===================================================================
File: hello.c                 Status: Up-to-date
   Working revision:  1.5.2.1 Wed May  5 00:13:58 1999
   Repository revision:       1.5.2.1 /usr/local/cvs/miproyecto/hello.c,v
   Sticky Tag:                Release-1999_05_01-bugfixes (branch: 1.5.2)
   Sticky Date:               (none)
   Sticky Options:            (none)
===================================================================
File: random.c                Status: Up-to-date
   Working revision:  1.2.2.1 Wed May  5 00:14:25 1999
   Repository revision:       1.2.2.1 /usr/local/cvs/miproyecto/b-subdir/random.c,v
   Sticky Tag:                Release-1999_05_01-bugfixes (branch: 1.2.2)
   Sticky Date:               (none)
   Sticky Options:            (none)
floss$
```

Ahora tienen cuatro dígitos en lugar de dos!

Un vistazo más de cerca nos revela que el número de revisión de cada fichero es simplemente el número de la rama (tal como se indica en la línea `Sticky Tag`), con un dígito extra al final.

Lo que está presenciando es tan sólo una parte del funcionamiento interno de CVS. Aunque casi siempre utilizará una rama para marcar una divergencia que afecte a la globalidad del proyecto, en realidad CVS registra la rama de forma individual, fichero a fichero. Este proyecto tenía cinco ficheros en el momento de crear la rama, así que en realidad se han creado cinco ramas, todas ellas con la misma marca: `Release-1999_05_01-bugfixes`.

La mayoría de la gente considera esta forma de hacer las cosas como una implantación bastante poco elegante por parte de CVS, pero en realidad lo que estamos viendo aquí es parte del legado de RCS: RCS no sabía cómo agrupar ficheros en los proyectos, y a pesar de que CVS sí lo hace, sigue utilizando código heredado de RCS para manejar las ramas.

Por regla general, usted no necesitará preocuparse demasiado por cómo CVS registra las cosas de forma interna, pero en este caso, resulta útil comprender la relación que existe entre números de ramas y números de revisiones. Veamos el fichero hello.c; todo lo que estoy a punto de decirle sobre hello.c se aplica a cualquier otro fichero presente en la rama, cambiando los números de revisión y de rama según convenga.

En el momento del nacimiento de la rama, el fichero hello.c se encontraba en su revisión 1.5. Cuando creamos la rama, se añadió un nuevo número al final para así formar un número de rama (CVS elige el primer número entero par que no sea cero y que esté libre). Por tanto, en este caso, el número de rama terminó siendo `1.5.2`. El número de la rama no es en sí mismo un número de revisión, pero sí es la raíz (es decir, el prefijo) de todos los números de revisión para hello.c que se emplearán en esta rama.

Sin embargo, cuando ejecutamos aquel primer comando "CVS status" en una copia de trabajo ramificada, el número de revisión de hello.c apareció como `1.5` solamente, y no como `1.5.2.0` o algo parecido. Esto se debe a que la revisión inicial de una rama es siempre la misma que la revisión que el fichero tiene en el tronco, donde nació la rama. Por tanto, CVS mostrará el número de revisión del tronco en el informe de estado mientras el fichero sea el mismo tanto en la rama como en el tronco.

Una vez que enviamos una nueva revisión al repositorio, hello.c ya no era igual en el tronco que en la rama: la copia que estaba en la rama había cambiado, mientras que la copia presente en el tronco seguía igual. Es por ello por lo que se asignó a hello.c su primer número de revisión de rama, tal como pudimos comprobar después de hacer el envío en el informe de estado, donde su número de revisión aparecía claramente como `1.5.2.1`.

Esta misma situación se aplica al fichero random.c. Su número de revisión en el momento de crear la rama era `1.2`, así que su primera rama es `1.2.2`, y el primer nuevo envío de random.c en esta rama recibió el número de revisión `1.2.2.1`.

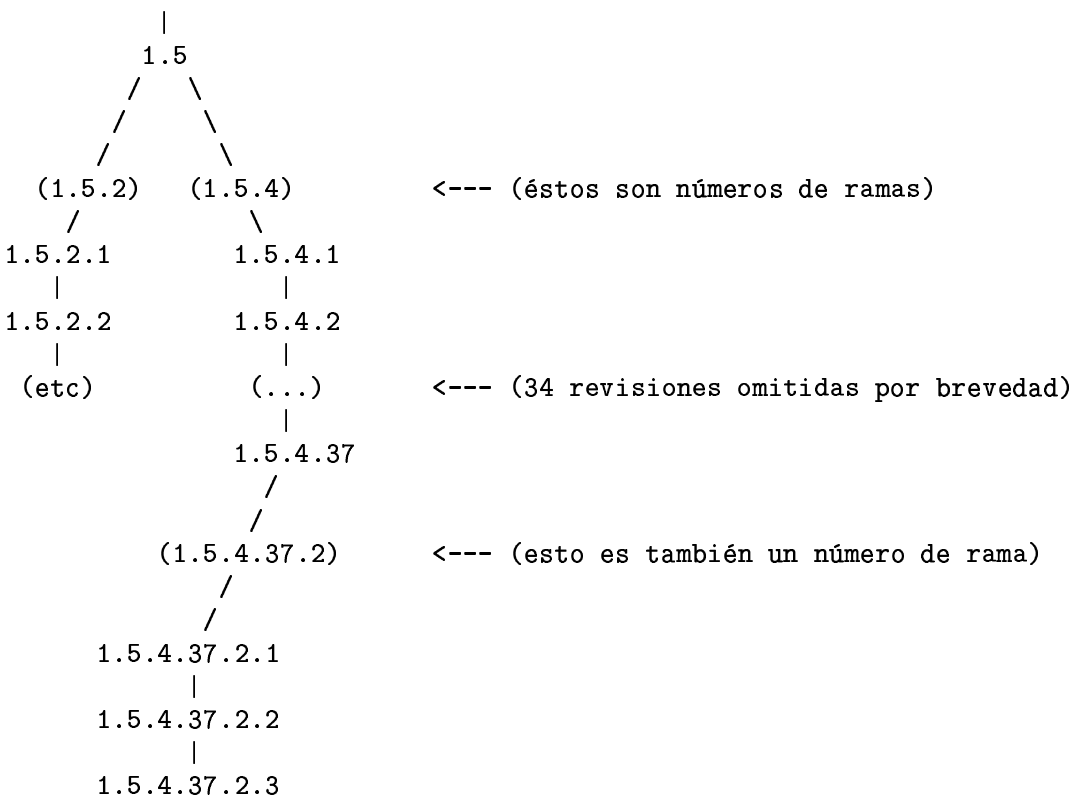
No existe ninguna relación numérica entre `1.5.2.1` y `1.2.2.1`. No hay razón alguna para pensar que forman parte de la misma rama excepto por el hecho de que ambos ficheros están marcados con `Release-1999_05_01-bugfixes`, y que la marca está asociada a los números de rama `1.5.2` y `1.2.2` en los respectivos ficheros. Por tanto, el nombre de la marca es su único recurso en la rama para concebirla como una entidad global. Aunque es perfectamente posible trasladar un fichero a una rama usando directamente el número de revisión,

```
floss$ cvs update -r 1.5.2.1 hello.c
U hello.c
floss$
```

casi siempre es una mala idea hacerlo, puesto que estaría mezclando la revisión en la rama de un fichero con las revisiones fuera de rama de otros. Quién sabe qué ficheros podría perder? Es mejor usar la marca de la rama para referirse a la rama y tratar todos los ficheros de una sola vez, evitando referirnos a ningún fichero en concreto; de esta forma no tiene que conocer ni preocuparse del número de revisión de rama de ningún fichero en particular.

También es posible hacer ramas que nacen de otras ramas, hasta llegar a niveles que podrían considerarse absurdos. Por ejemplo, si un fichero tuviese el número de revisión `1.5.4.37.2.3`, el historial de sus revisiones podría esquematizarse con algo como esto:

```
1.1
 |
1.2
 |
1.3
 |
1.4
```



[Figura 2.4: Un número extrañamente elevado de ramificaciones. El tiempo transcurre hacia abajo.]

Naturalmente, sólo circunstancias muy especiales harían necesario tal grado de ramificaciones, pero, no es agradable saber que CVS llegará todo lo lejos que usted se proponga? Las ramas anidadas se crean de la misma forma que cualquier otra rama: obtenga una copia de trabajo de la rama `N`, ejecute "cvs tag -b nombre_de_rama" sobre ella, y de esta forma creará la rama `N.M` en el repositorio (donde `N` representa el número de revisión de rama apropiado en cada fichero, como por ejemplo `1.5.2.1`, mientras que `M` representa la siguiente rama disponible al final de ese número, como por ejemplo `2`).

Fusión de cambios desde las ramas al tronco

Ahora que hemos aplicado la corrección del fallo a la rama, sincronicemos la copia de trabajo con la revisiones más recientes presentes en el tronco y veamos si también allí es necesaria la corrección. Vamos a obtener la copia de trabajo desde la rama empleando "update -A" (en este aspecto, las marcas de rama son como cualquier otra propiedad pegadiza) y entonces ver las diferencias respecto a la rama que acabamos de dejar:

```
floss$ cvs -q update -d -A
U hello.c
U b-subdir/random.c
floss$ cvs -q diff -c -r Release-1999_05_01-bugfixes
```

```
Index: hello.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.5.2.1
retrieving revision 1.6
diff -c -r1.5.2.1 -r1.6
*** hello.c   1999/05/05 00:15:07     1.5.2.1
--- hello.c   1999/05/04 20:19:16     1.6
***************
*** 4,9 ****
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("entre el saludo y la despedida\n");
    printf ("Adiós, mundo!\n");
  }
--- 4,10 --
  main ()
  {
    printf ("Hola, mundo!\n");
!   printf ("entre hola y adiós\n");
    printf ("Adiós, mundo!\n");
+   /* un comentario en la última línea */
  }
Index: b-subdir/random.c
===================================================================
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.2.2.1
retrieving revision 1.2
diff -c -r1.2.2.1 -r1.2
*** b-subdir/random.c 1999/05/05 00:15:07     1.2.2.1
--- b-subdir/random.c 1999/04/19 06:35:27     1.2
***************
*** 4,8 ****
  void main ()
  {
!   printf ("Un número aleatorio.\n");
  }
--- 4,8 --
  void main ()
  {
!   printf ("un número aleatorio\n");
  }
floss$
```

El comando "diff" muestra que hay una línea que difiere en la revisión en rama de hello.c, y que la revisión del tronco de este fichero tiene un comentario cerca del final que la revisión de la rama no tiene. Mientras, en random.c, la revisión en la rama tiene una "U" mayúscula y un punto, mientras que en la revisión presente en el tronco no aparece esto.

Para fusionar los cambios de una rama con la actual copia de trabajo, haga una actualización con la opción -j (la misma j de "juntar" que utilizamos anteriormente para revertir un fichero a una revisión más antigua):

```
floss$ cvs -q update -d -j Release-1999_05_01-bugfixes
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.5
retrieving revision 1.5.2.1
Merging differences between 1.5 and 1.5.2.1 into hello.c
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.2
retrieving revision 1.2.2.1
Merging differences between 1.2 and 1.2.2.1 into random.c
floss$ cvs -q update
M hello.c
M b-subdir/random.c
floss$ cvs -q ci -m "fusión con la rama Release-1999_05_01-bugfixes"
Checking in hello.c;
/usr/local/cvs/miproyecto/hello.c,v  <-  hello.c
new revision: 1.7; previous revision: 1.6
done
Checking in b-subdir/random.c;
/usr/local/cvs/miproyecto/b-subdir/random.c,v  <-  random.c
new revision: 1.3; previous revision: 1.2
done
floss$
```

Este ejemplo toma los cambios habidos desde la raíz de la rama hasta su extremo final y más reciente, y los fusiona con la copia de trabajo actual, que a partir de ese momento mostrará esas mismas modificaciones como si sus ficheros hubieran sido editados a mano para llevarlos a ese estado. Los cambios son entonces aplicados al tronco, puesto que nada había cambiado en el repositorio cuando una copia de trabajo sufrió un proceso de fusión.

Si bien en este ejemplo no se han producido conflictos, es bastante posible (e incluso probable) que hubiera algunos en una fusión hecha en un proyecto real, en cuyo caso estos conflictos tendrían que resolverse igual que cualquier otro conflicto, para a continuación poder aplicar los cambios.

Fusiones múltiples

A veces, una rama seguirá teniendo un desarrollo activo aún después de que su contenido haya sido fusionado con el tronco. Por ejemplo, esto puede suceder si se descubre un segundo fallo en la antigua distribución pública y este fallo ha de ser corregido en la rama. Cabe la posibilidad de que alguien no hubiese entendido la broma que hay en random.c, así que tendría usted que añadir una línea explicándola:

```
floss$ pwd
/home/loquesea/miproyecto_rama
floss$ cat b-subdir/random.c
/* Imprimir un número aleatorio. */
#include <stdio.h>
void main ()
```

```
{
  printf ("Un número aleatorio.\n");
  printf ("Ha entendido el chiste?\n");
}
floss$
```

y enviar el cambio. Si también es necesario fusionar esa corrección en el tronco, podría tener la tentación de utilizar el mismo comando "update" de antes en la copia de trabajo presente en el tronco para llevar a cabo la "re-fusión":

```
floss$ cvs -q update -d -j Release-1999_05_01-bugfixes
RCS file: /usr/local/cvs/miproyecto/hello.c,v
retrieving revision 1.5
retrieving revision 1.5.2.1
Merging differences between 1.5 and 1.5.2.1 into hello.c
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.2
retrieving revision 1.2.2.2
Merging differences between 1.2 and 1.2.2.2 into random.c
rcsmerge: warning: conflicts during merge
floss$
```

Como puede ver, este comando no ha tenido el efecto deseado: nos encontramos con un conflicto, a pesar de que la copia en el tronco no había sido modificada y, por tanto, no esperábamos encontrarnos ninguno.

El problema reside en que el comando "update" se ha comportado exactamente de la forma descrita: ha intentado tomar todos los cambios habidos desde la raíz de la rama y su extremo final, y a continuación fusionarlos tomando como referencia la copia de trabajo actual. El único problema está en que algunos de estos cambios ya habían sido fusionados desde esta copia de trabajo, de ahí que surgiese el conflicto:

```
floss$ pwd
/home/loquesea/miproyecto
floss$ cat b-subdir/random.c
/* Imprimir un número aleatorio. */
#include <stdio.h
void main ()
{
<<<<<<< random.c
  printf ("Un número aleatorio.\n");
=======
  printf ("Un número aleatorio.\n");
  printf ("Ha entendido el chiste?\n");
>>>>>>> 1.2.2.2
}
floss$
```

Llegados a este punto, podría intentar resolver estos conflictos a mano, dado que no es difícil ver lo que es necesario hacer en cada fichero. Sin embargo, es todavía mejor tomar medidas desde el principio para evitar conflictos. Pasando dos opciones "-j" en lugar de una, obtendrá sólo los cambios habidos desde la última vez que hizo una fusión con el extremo final de la rama, en lugar de tener en consideración todos los cambios habidos en ella. La

primera -j le da el punto inicial de la rama, y la la segunda es sólo el nombre de la rama, que implica su extremo final y más reciente.

La cuestión entonces es, cómo puede especificar el punto de la rama desde el que quizo la última fusión? Una forma de hacerlo sería indicar una fecha junto con el nombre de la marca dispuesta en la rama. CVS ofrece para ello una sintaxis especial:

```
floss$ cvs -q update -d -j "Release-1999_05_01-bugfixes:2 days ago" \
                        -j Release-1999_05_01-bugfixes
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.2.2.1
retrieving revision 1.2.2.2
Merging differences between 1.2.2.1 and 1.2.2.2 into random.c
floss$
```

Si el nombre de la rama va seguido de un signo de dos puntos y a continuación por una fecha en un formato válido para CVS, CVS incluirá solamente los cambios habidos después de esa fecha. De este modo, si sabe que la corrección original del fallo se envió a la rama hace tres días, el comando precedente fusionaría solamente la segunda corrección.

Una forma mejor de hacer esto, en este caso adelantándonos para tener en cuenta este tipo de eventualidades, sería marcar la rama después de implantar cada corrección (sólo una marca normal; no se trata de iniciar una nueva rama ni nada parecido). Supongamos que después de corregir el fallo en la rama y aplicar la corrección al repositorio, hacemos esto en la copia de trabajo de la rama:

```
floss$ cvs -q tag Release-1999_05_01-bugfixes-correc1
T README.txt
T hello.c
T a-subdir/loquesea.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
floss$
```

De esa forma, cuando llegue el momento de fusionar el segundo cambio en el tronco, podrá utilizar la marca que ha tenido la buena idea de colocar ahí para delimitar la revisión anterior:

```
floss$ cvs -q update -d -j Release-1999_05_01-bugfixes-correc1 \
                        -j Release-1999_05_01-bugfixes
RCS file: /usr/local/cvs/miproyecto/b-subdir/random.c,v
retrieving revision 1.2.2.1
retrieving revision 1.2.2.2
Merging differences between 1.2.2.1 and 1.2.2.2 into random.c
floss$
```

Ni que decir tiene que este método es mucho mejor que intentar recordar cuánto tiempo hace que hizo tal o cual modificación, pero sólo funcionará si se acuerda de marcar la rama cada vez que la fusione con el tronco. La lección aprendida aquí es, por tanto, marcar pronto y a menudo! Es mejor pecar de tener demasiadas marcas (siempre y cuando tengan nombres lo suficientemente descriptivos) que de tener muy pocas. En estos últimos ejemplos que le he dado no había ninguna necesidad de que la nueva marca de la rama tuviese un nombre similar al de la marca de la rama en sí. Si bien yo utilicé el nombre `Release-1999_05_01-bugfixes-correc1`, también podría haber sido `correc1`; sin embargo, es preferible

emplear el primero, dado que contiene el nombre de la rama y por tanto no existirá la posibilidad de que sea confundido con la marca de alguna otra rama. (Recuerde que los nombres de las marcas son únicos dentro de los ficheros, no dentro de las ramas. No puede tener dos marcas llamadas `correc1` en el mismo fichero, aunque se refieran a revisiones que se encuentran en diferentes ramas.)

Crear una marca o rama sin una copia de trabajo

Como ya hemos dicho, marcar es algo que afecta al repositorio, no a la copia de trabajo. Lo cual da pie a que nos preguntemos: por qué entonces es necesario disponer de una copia de trabajo para marcar?

Pues bien, el único fin que cumple es el de designar qué proyecto y qué revisiones en concreto de los ficheros del proyecto se están marcando. Si pudiera especificar el proyecto y las revisiones independientemente de la copia de trabajo, no sería necesario disponer de ésta.

Y mire usted por dónde, hay una manera de hacer esto: el comando "rtag" (de "repository tag", o marca de repositorio, N. del T.). Es muy similar a "tag"; un par de ejemplos bastarán para explicar cómo se usa. Volvamos atrás al momento en el que llegó a nuestro buzón el primer informe de fallo y necesitábamos crear una rama que partiese de la última distribución pública. En nuestro caso, lo que hicimos fue solicitar una copia de trabajo con la marca de distribución y después ejecutar `tag -b` sobre ella:

```
floss$ cvs tag -b Release-1999_05_01-bugfixes
```

Esto creó una rama que partía de `Release-1999_05_01`. Sin embargo, puesto que conocemos la marca de la distribución, podríamos haberla usado en un comando "rtag" para indicar dónde plantar la rama exactamente, sin ni siquera procurarnos antes una copia de trabajo:

```
floss$ cvs rtag -b -r Release-1999_05_01 Release-1999_05_01-bugfixes miproyecto
```

Y éso es todo. Este comando puede darse desde cualquier lugar, sea dentro o fuera de una copia de trabajo. Sin embargo, su variable de entorno CVSROOT debe apuntar, por supuesto, al repositorio, o bien puede referirse a él empleando la opción global -d. También funciona para marcas que no sean de ramas, pero es menos útil de esta forma ya que tendrá que indicar el número de revisión de cada fichero, uno por uno. (O puede referirse a ellos empleando una marca, pero entonces obviamente ya tendría una marca ahí, en cuyo caso, para qué iba a querer poner una segunda marca a ese mismo grupo de revisiones?)

Ahora ya sabe lo suficiente como para manejarse con CVS, y posiblemente lo bastante como para empezar a trabajar con otras personas en un proyecto. Todavía quedan algunas características de menos importancia que no se han abordado, así como algunas opciones para los comandos ya comentados, que también resultan útiles. Todas ellas serán presentadas donde corresponda a lo largo de los próximos capítulos, en situaciones que le mostrarán cómo y por qué utilizarlas. Cuando tenga alguna duda, no dude en consultar el manual Cederqvist, un recurso indispensable para los usuarios habituales de CVS.

Administracion del Repositorio

En ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩, usted ha aprendido bastante de CVS para usarlo de forma eficaz como participante de un proyecto. Sin embargo, si va a ser administrador de un proyecto, necesitará conocer cómo instalar CVS y administrar repositorios. En este capítulo descorreremos la cortina y observaremos en detalle cómo se estructura el repositorio, y cómo lo utiliza CVS. Aprenderá los pasos más importantes por los que pasa CVS durante las actualizaciones y envíos ("commits"), y cómo puede modificar su comportamiento. Comprendiendo cómo trabaja CVS, será también capaz de encontrar las causas de los problemas, y resolverlos de forma mantenible.

Esto puede parecer muy complicado, pero recuerde que CVS ya ha probado ser bastante duradero, y seguramente seguirá usándose durante muchos años más. Lo que aprenda ahora le será de utilidad durante mucho tiempo. CVS también tiende a hacerse más indispensable cuanto más lo use. Si va a depender tanto de algo (y hágame caso, así será), realmente vale la pena que llegue a conocerlo.

Con esto en mente, vamos a comenzar por el principio: poniendo CVS en su sistema.

Consiguiendo e instalando CVS

En muchos casos, no tendrá que salir a buscar CVS, porque ya estará en su sistema. Si posee una de las distribuciones principales de Linux o FreeBSD, seguramente está instalado en /usr/bin o alguna otra localización probable. Si no, los usuarios de Red Hat Linux pueden encontrar por lo general un RPM ("Red Hat Package", o Paquete de Red Hat) con la última versión de CVS (o casi la última) en sus distribuciones. Y los usuarios de Debian pueden instalar el último paquete Debian con estas órdenes:

```
floss$ apt-get update
floss$ apt-get install cvs
```

Si CVS no está ya en su máquina, probablemente tendrá que compilarlo a partir del código fuente. Si no es un usuario de Unix, seguramente encuentre más fácil conseguir un binario precompilado para su sistema operativo (más adelante se detalla este aspecto). Afortunadamente, CVS está totalmente *autoconfigurado* – es decir, utiliza el mecanismo de autoconfiguración de GNU, con lo que la compilación a partir de las fuentes es sorprendentemente sencilla.

Consiguiendo y compilando CVS bajo Unix

En el momento de escribir esto, existen dos sitios principales de los que se puede descargar CVS. Uno es el servidor FTP de la Fundación para el Software Libre, `ftp://ftp.gnu.org/gnu/cvs/`, que ofrece CVS como herramienta oficial GNU. El otro es el sitio de descarga de Cyclic Software. Cyclic Software es, si no el "administrador" de CVS, sí el "administrador de los administradores", proporcionando un servidor de repositorio y acceso para descargar a usuarios y desarrolladores. Distribuyen versiones desde `http://download.cyclic.com/pub/`.

Cualquiera de los dos sitios es bueno. En el siguiente ejemplo, uso el sitio de Cyclic Software. Si dirige su cliente FTP (seguramente su navegador Web) hacia allí, verá una lista de directorios, algo como esto:

```
Index of /pub
    cvs-1.10.5/            18-Feb-99 21:36      -
    cvs-1.10.6/            17-May-99 10:34      -
    cvs-1.10/              09-Dec-98 17:26      -
    macintosh/             23-Feb-99 00:53      -
    os2/                   09-Dec-98 17:26      -
    packages/              09-Dec-98 17:26      -
    rcs/                   09-Dec-98 17:26      -
    tkcvs/                 09-Dec-98 17:26      -
    training/              09-Dec-98 17:26      -
    unix/                  09-Dec-98 17:26      -
    vms/                   09-Dec-98 17:26      -
```

Preste atención a los directorios que empiezan por "cvs-" (puede ignorar la mayoría de los demás). Hay tres directorios de este tipo, lo que significa que se enfrenta a una elección: Descargar la versión llamada "estable", o ir a por una versión intermedia más reciente (pero menos probada). Las versiones estables tienen sólo un punto decimal, como en "cvs-1.10", mientras que las versiones intermedias tienen incrementos de versión menores añadidos al final, como en "1.10.5".

El sitio de GNU sólo ofrece las versiones principales, no las intermedias, así que no verá todas las anteriores si consigue CVS desde aquí. En general, las versiones intermedias han sido bastante seguras, y a veces resuelven problemas encontrados en las versiones principales. La mejor política es ir a por la versión intermedia más alta, pero si encuentra cualquier problema con ella, prepárese para bajar a la versión anterior, tantas veces como sea necesario. La versión más alta listada en el ejemplo anterior es cvs-1.10.6. Entrando en ese directorio, veremos esto:

```
Index of /pub/cvs-1.10.6
    cvs-1.10.6.tar.gz      17-May-99 08:44   2.2M
```

Esto es – el código fuente completo de CVS. Descárguelo a su máquina y estará preparado para compilar. En este punto, si ya esta familiarizado con el proceso de compilación estándar para herramientas GNU, sabrá qué hacer y probablemente no necesite leer nada desde aquí a la sección ⟨undefined⟩ [Anatomia de una distribucion CVS], page ⟨undefined⟩. Por otra parte, si no está seguro de cómo continuar, siga leyendo....

Los siguientes ejemplos e instrucciones de compilación asumen que posee una distribución estándar de Unix. Cualquiera de las versiones libres de Unix (por ejemplo, FreeBSD o Linux) debería funcionar sin problemas, como debería ocurrir en las principales versiones comerciales de Unix (como SunOS/Solaris, AIX, HP-UX, o Ultrix). Incluso si estas instrucciones no funcionan para usted exactamente como están escritas, no renuncie a la esperanza. Aunque cubrir los detalles de compilar en cada sistema operativo está fuera de los objetivos de este libro, daré indicaciones de otras fuentes de ayuda más adelante en este capítulo.

De todas formas, para seguir adelante con la compilación, primero descomprima el fichero tar usando GNU unzip y tar (si no los tiene instalados en su sistema, puede conseguir gunzip de `ftp://ftp.gnu.org/gnu/gzip/` y la versión GNU de tar de `ftp://ftp.gnu.org/gnu/tar/`):

```
floss$ gunzip cvs-1.10.6.tar.gz
floss$ tar xvf cvs-1.10.6.tar
```

Verá muchos nombres de fichero volando por su pantalla.

Ahora tendrá un nuevo directorio en su máquina – cvs-1.10.6 –, que contendrá el código fuente de CVS. Entre al directorio y configure CVS para su sistema, usando el guión "configure" proporcionado:

```
floss$ cd cvs-1.10.6
floss$  ./configure
creating cache ./config.cache
checking for gcc... gcc
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking how to run the C preprocessor... gcc -E
  (etc)
```

Cuando la orden "configure" finalice, el árbol fuente sabrá todo lo que necesita saber para compilarse en su máquina. El siguiente paso consiste en escribir:

```
floss$ make
```

Verá muchas líneas de salida en su pantalla, después escriba:

```
floss$ make install
```

Verá aún más líneas de salida volando; cuando haya acabado del todo, CVS estará instalado en su sistema. (Seguramente necesitará hacer este último paso como superusuario.)

Por defecto, el ejecutable de CVS acabará siendo '`/usr/local/bin/cvs`'. Esto asume que usted tiene un programa "make" decente instalado en su sistema (de nuevo, si no tiene uno, consiga el "make" del proyecto GNU en `ftp://ftp.gnu.org/gnu/make/`).

Si quiere que CVS se instale en una ruta distinta de /usr/local/bin, debería cambiar la forma en que ejecuta el paso inicial de configuración. Por ejemplo,

```
floss$ ./configure --prefix=/usr
```

da lugar a que CVS esté instalado como /usr/bin/cvs (siempre acaba en PREFIX/bin/cvs). El prefijo por defecto es /usr/local, que está bien para la mayoría de las instalaciones.

Nota Para Usuarios Experimentados: Aunque las versiones antiguas de CVS consistían en más que un mero ejecutable puesto que dependían de tener instalado RCS también, este no ha sido el caso desde la Versión 1.10. Por ello, no necesita preocuparse por ninguna librería o ejecutable aparte de cvs mismo.

Si solamente pretende usar CVS para acceder a repositorios remotos, lo anterior es todo lo que necesita hacer. Si además planea servir un repositorio desde su sistema, serán necesarios algunos pasos adicionales que se explican más adelante en este capítulo.

Consiguiendo e instalando CVS bajo Windows

A menos que sea realmente fanático respecto a tener el código fuente de su ejecutable, no necesitará compilar CVS a partir de las fuentes en su caja Windows. Al contrario que en Unix, probablemente las herramientas necesarias para compilar no existan en su sistema, por lo que una compilación implicaría primero conseguir estas herramientas. Dado que eso está fuera de los objetivos de este libro, simplemente daré unas instrucciones para conseguir un binario de CVS precompilado.

Antes de nada hay que notar que las distribuciones binarias de CVS para Windows generalmente se hacen sólo para las versiones principales de CVS – no para las

intermedias – y no se encuentran en el sitio FTP de GNU. Así que necesitará ir al sitio de descarga de Cyclic Software, donde en el directorio de la versión principal, `http://download.cyclic.com/pub/cvs-1.10/`, verá un subdirectorio adicional

```
Index of /pub/cvs-1.10
    cvs-1.10.tar.gz        14-Aug-98 09:35   2.4M
    windows/
```

dentro del cual hay un fichero ZIP:

```
Index of /pub/cvs-1.10/windows
    cvs-1.10-win.zip       14-Aug-98 10:10   589k
```

Este fichero ZIP contiene una distribución binaria de CVS. Descárguela y descomprima este fichero ZIP:

```
floss$ unzip cvs-1.10-win.zip

Archive:  cvs-1.10-win.zip
  inflating: cvs.html
  inflating: cvs.exe
  inflating: README
  inflating: FAQ
  inflating: NEWS
  inflating: patch.exe
  inflating: win32gnu.dll
```

El fichero README contiene instrucciones detalladas. Para la mayoría de las instalaciones, pueden resumirse como sigue: Ponga todos los ficheros EXE y DLL en un directorio incluido en su PATH. Además, si va a usar el método pserver para acceder a un repositorio remoto, quizá necesite añadir lo siguiente a su fichero '`C:\AUTOEXEC.BAT`' y reiniciar:

```
set HOME=C:
```

Esto le dice a CVS dónde almacenar el fichero .cvspass.

Actualmente CVS bajo Windows no puede servir repositorios a máquinas remotas; puede ser un cliente (conectándose a repositorios remotos), y operar en modo local (usando un repositorio en la misma máquina). Por lo general, este libro asume que CVS bajo Windows funciona como cliente. Sin embargo, no debería ser demasiado difícil poner a punto un repositorio local bajo Windows después de leer las instrucciones orientadas a Unix del resto de este capítulo.

Si sólo accede a repositorios remotos, puede que ni siquiera necesite ejecutar CVS. Existe una utilidad llamada WinCvs que implementa sólo la parte cliente de CVS. Se distribuye por separado de CVS pero, como CVS, está disponible libremente bajo la Licencia Pública General (GPL) de GNU. Hay más información disponible en `http://www.wincvs.org`.

Consiguiendo e instalando CVS en un Macintosh

CVS está disponible para Macintosh, pero no como parte de la distribución principal. De momento, hay realmente disponibles tres clientes CVS para Macintosh por separado:

MacCvs – `http://www.wincvs.org`

MacCVSClient – `http://www.glink.net.hk/~jb/MacCVSClient` o `http://www.cyclic.com/maccvsclient/`

MacCVS Pro – `http://www.maccvs.org`

Francamente, no sé cuál es el mejor. Pruébelos todos, no necesariamente en el orden dado, y vea cuál le gusta. MacCVS Pro parece tener un desarrollo activo. MacCvs aparentemente es un proyecto compañero de WinCVS y comparte página web con él. (En el momento de escribir esto, un aviso en la página de WinCVS afirma: "El desarrolo de MacCvs se reanudará pronto.", ignoro lo que signifique esto.)

Limitaciones de las versiones Windows y Macintosh

Las distribuciones de CVS para Windows y Macintosh están por lo general limitadas en funcionalidad. Todas pueden actuar como clientes, en el sentido de que pueden contactar con un servidor de repositorio para obtener una copia válida, enviar cambios, actualizar, etc. Pero no pueden servir repositorios por ellas mismas. Si la configura correctamente, la versión de Windows podrá usar un repositorio en un disco local, pero aún no podrá servir proyectos desde ese repositorio a otras máquinas. En general, si desea tener un repositorio CVS accesible por red, deberá ejecutar el servidor CVS en una máquina Unix.

Anatomia de una distribucion CVS

Las instrucciones anteriores están diseñadas para ponerle en marcha rápidamente, pero hay mucho más dentro de una distribución de fuentes de CVS que simplemente el código. Aquí tiene un repaso rápido al árbol fuente, de forma que sepa qué partes son recursos útiles y cuáles pueden ignorarse.

Ficheros informativos

En el nivel superior del árbol de la distribución, encontrará algunos ficheros que contienen información útil (y enlaces a información más detallada). Estos ficheros son, en orden de importancia aproximado:

'`NEWS`' – Este fichero enumera los cambios de una versión a la siguiente, en orden cronológico inverso (es decir, la más reciente al principio). Si ya ha estado usando CVS durante un tiempo y se ha actualizado a una nueva versión, debería mirar en el fichero NEWS para ver qué nuevas funciones están disponibles. Además, aunque la mayoría de los cambios de CVS conservan compatibilidad hacia atrás, de vez en cuando hay cambios no compatibles. Es mejor leer algo acerca de ellos aquí que sorprenderse cuando CVS no se comporte como espera que lo haga.

'`BUGS`' – Este fichero contiene exactamente lo que piensa: una lista de errores conocidos en CVS. Generalmente no harán que la versión de CVS sea inutilizable, pero debería leer el fichero cada vez que instale una nueva versión.

'`DEVEL-CVS`' – Este fichero es la "constitución" de CVS. Describe el proceso mediante el cual se aceptan los cambios en la distribución principal de CVS y los procedimientos mediante los que una persona se convierte en desarrollador de CVS. No necesita leerlo si sólo quiere usar CVS; sin embargo, resulta muy interesante si quiere comprender cómo los esfuerzos descoordinados de gente repartida por todo el mundo convergen en un programa que funciona. Y por supuesto, hace falta leerlo si planea enviar un parche (sea para reparar un error o para una nueva característica) a CVS.

'`HACKING`' – Pese a su nombre, el fichero HACKING no dice mucho del diseño o implementación de CVS. Es principalmente una guía de estándares de programación y otros asuntos técnicos para gente que piense en escribir un parche para CVS. Puede verse como un apéndice del fichero DEVEL-CVS. Después de comprender la filosofía básica del desarrollo de CVS, debe leer el fichero HACKING para traducir esta filosofía a unas prácticas concretas de programación.

'`FAQ`' – Éste es el documento de "Preguntas Frecuentes" (FAQ o "Frequently Asked Questions" en inglés). Desgraciadamente ha tenido una historia de mantenimiento bastante irregular. David Grubbs se encargó de él hasta 1995. En ese momento (presumiblemente) estaba muy ocupado, y el fichero languideció durante un tiempo. Finalmente, in 1997, Pascal Molli retomó el mantenimiento. Molli tampoco tuvo tiempo para mantenerlo a mano, pero al menos encontró tiempo para ponerlo en su sistema automático FAQ-O-Matic, que permite al público mantener las FAQ de un modo descentralizado (básicamente, cualquiera puede editar o añadir entradas mediante un formulario Web). Probablemente esto fue una buena idea, puesto que al menos las FAQ están siendo actualizadas de nuevo; sin embargo, la organización general y el control de calidad no son del mismo nivel que si una única persona las mantuviera.

La versión maestra de las FAQ está siempre disponible en el sitio Web de Molli (`http://www.loria.fr/~molli/cvs-index.html`, bajo el enlace "Documentation"). El fichero FAQ que se encuentra en las distribuciones CVS se genera automáticamente desde la base de datos de FAQ-O-Matic, así que para cuando llega al público está un poco anticuado. Sin embargo, puede ser de gran ayuda si busca sugerencias y ejemplos sobre cómo hacer algo específico (como fusionar una rama enorme al tronco o "resucitar" un fichero eliminado). La mejor forma de usarlo es como documento de referencia; puede abrirlo en su editor favorito y hacer búsquedas de los términos que le interesen. Intentar usarlo como un tutorial sería un error – le faltan demasiados puntos importantes sobre CVS para servir de guía completa.

Subdirectorios

La distribución CVS contiene unos cuantos subdirectorios. En el curso de una instalación normal no tendrá que navegar por ellos, pero si quiere curiosear por las fuentes, está bien saber qué hace cada cosa. Aquí están:

```
contrib/
diff/
doc/
emx/
lib/
man/
os2/
src/
tools/
vms/
windows-NT/
zlib/
```

La mayoría de ellos pueden ignorarse. Los subdirectorios emx/, os2/, vms/, and windows-NT/ contienen código fuente específico del sistema operativo, por lo que usted

sólo los necesitaría si realmente está tratando de arreglar un problema a nivel de código en CVS (una situación improbable, aunque no sería la primera vez). Los subdirectorios diff/ and zlib/ contienen implementaciones internas de CVS del programa diff y de la biblioteca de compresión GNU zip, respectivamente. (CVS usa la última para reducir el número de bits que tiene que enviar por la red cuando accede a repositorios remotos.)

Los subdirectorios contrib/ y tools/ contienen programas libres hechos por terceros para usarse con CVS. En contrib/, encontrará una ordenación de guiones de consola ("shell scripts", N. del T.) pequeños y especializados (lea contrib/README para averiguar lo que hacen). El subdirectorio tools/ solía contener contribuciones, pero ahora contiene un fichero README que dice en parte:

```
Este subdirectorio antiguamente contenía herramientas que pueden usarse
con CVS.  En particular, solía contener una copia de la versión 1.x de
pcl-cvs. Pcl-cvs es una interfaz de Emacs para CVS.

Si está buscando pcl-cvs le sugerimos la versión 2.x de pcl-cvs en:
    ftp://ftp.weird.com/pub/local/
```

El paquete PCL-CVS a que se refiere es muy práctico, y tengo más que decir sobre él en ⟨undefined⟩ [Herramientas de terceros], page ⟨undefined⟩.

Los subdirectorios src/ y lib/ contienen el grueso del código fuente de CVS, incluido el codigo interno de CVS. Las principales estructuras de datos y órdenes están implementados en src/, mientras que lib/ contiene pequeños módulos de código de utilidad general que usa CVS.

El subdirectorio man/ contiene las páginas man de CVS (para el sistema de manual en línea de Unix). Cuando ejecutó "make install", se incorporaron dentro de las páginas man de su sistema Unix, así que puede escribir

```
floss$ man cvs
```

y conseguir una introducción y referencia de subórdenes (algo concisas) para CVS. Aunque son útiles como referencia rápida, las páginas del manual pueden no estar tan actualizadas ni ser tan completas como el manual Cederqvist (vea la próxima sección); sin embargo, si sirve de consuelo, es más probable que las páginas del manual estén incompletas que realmente sean incorrectas.

El manual Cederqvist

Con esto llegamos al subdirectorio doc/, cuyo habitante más importante es el famoso *Cederqvist*. Hoy en día seguramente sea excesivo llamarlo "el Cederqvist". Aunque Per Cederqvist (de Signum Support, Linkoping Suecia, www.signum.se) escribió la primera versión alrededor de 1992, muchas otras personas lo han actualizado desde entonces. Por ejemplo, cuando los desarrolladores añaden una nueva característica a CVS, generalmente también la documentan en el Cederqvist.

El Manual Cederqvist está escrito en formato Texinfo, usado por el proyecto GNU porque es relativamente fácil producir salidas tanto en línea como impresas a partir de él (en los formatos Info y PostScript respectivamente). El fichero maestro Texinfo es doc/cvs.texinfo, pero las distribuciones CVS vienen con los ficheros Info y PostScript pregenerados, así que no tiene que preocuparse de ejecutar herramientas Texinfo.

Aunque el Cederqvist puede usarse como introducción y tutorial, probablemente sea más útil como documento de referencia. Por esa razón, la mayoría de la gente navega por él en línea en lugar de imprimirlo (aunque el fichero PostScript es '`doc/cvs.ps`', para aquellos que tengan papel de sobra). Si ésta es la primera vez que ha instalado CVS en su sistema, tendrá que hacer un paso extra para asegurarse de que el manual está accesible en línea.

Los ficheros Info (doc/cvs.info, doc/cvs.info-1, doc/cvs.info-2, etc.) se instalaron cuando ejecutó "make install". Aunque los ficheros se copiaron en el árbol Info del sistema, puede que aún tenga que añadir una línea para CVS en la tabla de contenidos de Info, el nodo "Top". (Esto sólo será necesario si es la primera vez que ha instalado CVS en su sistema; de lo contrario, la entrada de instalaciones anteriores debería estar ya en la tabla de contenidos.)

Si ha añadido nueva documentación Into antes, puede que esté familiarizado con el proceso. Primero averigüe dónde se instalaron las páginas Info. Si usó la instalación por defecto (en /usr/local/), entonces los ficheros Info son /usr/local/info/cvs.info*. Si al instalar usó

```
floss$ ./configure --prefix=/usr
```

los ficheros acabaron siendo /usr/info/cvs.*. Después de localizar los ficheros, necesitará añadir una línea para CVS en la tabla de contenidos de Info, que está en un fichero llamado dir en ese directorio (así que en el último caso sería /usr/info/dir). Si no tiene acceso como root pídale a su administrador que lo haga. Aquí hay un extracto de dir antes de añadir la referencia a la documentación CVS:

```
* Bison: (bison).         The Bison parser generator.
* Cpp: (cpp).             The GNU C preprocessor.
* Flex: (flex).           A fast scanner generator
```

Y aquí está la misma zona de dir despues:

```
* Bison: (bison).         The Bison parser generator.
* Cpp: (cpp).             The GNU C preprocessor.
* Cvs: (cvs).             Concurrent Versions System
* Flex: (flex).           A fast scanner generator
```

El formato de la línea es muy importante. Debe incluir el asterisco, los espacios y los dos puntos en '`* Cvs:`', y los paréntesis y el punto en '`(cvs).`' detrás de él. Si falta cualquiera de estos elementos, el formato del Info dir estará corrupto, y será incapaz de leer el Cederqvist.

Una vez que el manual esté instalado y referenciado desde la tabla de contenidos, podrá leerlo con cualquier navegador compatible con Info. Los que estarán instalados con mayor seguridad en un sistema Unix típico son el lector Info de línea de órdenes, que puede invocarse así si quiere ir directo a las páginas de CVS

```
floss$ info cvs
```

y el incluido en Emacs, que se invoca escribiendo

```
M-x info
```

o

```
C-h i
```

Tómese el tiempo necesario para conseguir poner a punto el Cederqvist correctamente en su sistema cuando instale CVS; acortará mucho el camino cuando tenga que buscar algo.

Otras fuentes de información

Además del Cederqvist, las FAQ, y los demás ficheros de la propia distribución, hay recursos de Internet dedicados a CVS. Si va a administrar un servidor CVS seguramente quiera unirse a la lista de correo info-cvs. Para suscribirse envíe un mensaje a `info-cvs-request@gnu.org` (la lista en sí es `info-cvs@gnu.org`). El tráfico puede ser de medio a alto, de unos 10 a 20 mensajes diarios, casi siempre preguntas buscando respuestas. La mayoría de ellas pueden borrarse sin leerlas (a no ser que quiera ayudar a la gente respondiendo a sus preguntas, que siempre está bien), pero de vez en cuando alguien anuncia el descubrimiento de un error, o un parche que implementa alguna característica que usted ha estado esperando.

También puede apuntarse a la lista de correo formal de informes de errores, que incluye todos los informes de error enviados. Probablemente no sea necesario, a menos que pretenda ayudar a arreglar los errores, que sería estupendo, o que sea terriblemente paranoico y quiera estar enterado de todos los problemas que otros encuentren con CVS. Si quiere apuntarse, envíe un mensaje a `bug-cvs-request@gnu.org`.

También hay un grupo de noticias de Usenet, `comp.software.config-mgmt`, en el que se habla de control de versiones y sistemas de administración de configuraciones en general, y donde hay mucha discusión acerca de CVS.

Por último, hay al menos tres sitios Web dedicados a CVS. El de Cyclic Software `http://www.cyclic.com` ha sido la página principal informal de CVS durante algunos años, y probablemente lo seguirá siendo en el futuro próximo. Cyclic Software también proporciona espacio en el servidor y acceso por red para el repositorio en el que se guardan las fuentes de CVS. Las páginas web de Cyclic contienen multitud de enlaces a parches experimentales para CVS, herramientas de terceros que trabajan con CVS, documentación, archivos de listas de correo, y todo lo demás. Si no puede encontrar lo que necesita en la distribución, `http://www.cyclic.com` es el lugar para empezar a buscar.

Otros dos buenos sitios son el de Pascal Molli `http://www.loria.fr/~molli/cvs-index.html` y el de Sean Dreilinger `http://durak.org/cvswebsites/`. La mayor atracción del sitio de Molli la forman, por supuesto, las FAQ, pero también tiene enlaces a herramientas relacionadas con CVS y archivos de listas de correo. El sitio de Dreilinger se especializa en información sobre el uso de CVS para administrar documentos Web y también tiene un motor de búsqueda específico para CVS.

Iniciando un repositorio

Una vez que el ejecutable CVS esté instalado en su sistema, podrá empezar a usarlo en seguida como cliente para acceder a repositorios remotos, siguiendo los procedimientos descritos en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩. Sin embargo, si quiere servir revisiones desde su máquina, tendrá que crear un repositorio en ella. La orden para hacerlo es

```
floss$ cvs -d /usr/local/nuevorepos init
```

donde '`/usr/local/nuevorepos`' es la ruta a donde usted quiera que esté el repositorio (por supuesto, deberá tener permiso de escritura en ese directorio, lo que podría implicar ejecutar la orden como root). En cierto modo puede parecer poco intuitivo que la local-

ización del repositorio nuevo se especifique antes de la suborden init en lugar de después de él, pero usando la opción -d sigue siendo consistente con otras órdenes CVS.

La orden acabará silenciosamente después de ejecutarse. Vamos a examinar el nuevo directorio:

```
floss$ ls -ld /usr/local/nuevorepos
drwxrwxr-x   3 root     root         1024 Jun 19 17:59 /usr/local/nuevorepos/
floss$ cd /usr/local/nuevorepos
floss$ ls
CVSROOT
floss$ cd CVSROOT
floss$ ls
checkoutlist     config,v        history     notify     taginfo,v
checkoutlist,v   cvswrappers     loginfo     notify,v   verifymsg
commitinfo       cvswrappers,v   loginfo,v   rcsinfo    verifymsg,v
commitinfo,v     editinfo        modules     rcsinfo,v
config           editinfo,v      modules,v   taginfo

floss$
```

El único subdirectorio del repositorio nuevo – CVSROOT/ – contiene varios ficheros de administración que controlan el comportamiento de CVS. Más adelante examinaremos esos ficheros uno a uno; por ahora, nuestro objetivo sólo es conseguir que el repositorio funcione. En este caso, "funcionar" significa que los usuarios puedan importar, actualizar, obtener copias de trabajo y enviar cambios a los proyectos.

No hay que confundir la variable de entorno CVSROOT introducida en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩ con este subdirectorio CVSROOT del repositorio. No tienen nada que ver – es una coincidencia desafortunada que compartan el mismo nombre. La primera es una forma de evitarles a los usuarios tener que teclear '`-d <situación-del-repositorio>`' cada vez que usen CVS; el segundo es el directorio de administración de un repositorio.

Una vez que el repositorio se haya creado, deberá ocuparse de sus permisos. CVS no requiere de ningún permiso estándar particular o sistema de propiedad de ficheros; simplemente necesita acceso de escritura al repositorio. Sin embargo – en parte por razones de seguridad, pero sobre todo por su propia salud como administrador – recomiendo encarecidamente que siga los siguientes pasos:

1. Añada un grupo de Unix `cvs` a su sistema. Cualquier usuario que necesite acceder al repositorio debería estar en el grupo. Por ejemplo, la línea del fichero '`/etc/group`' de mi máquina es:

   ```
   cvs:*:105:kfogel,sussman,jimb,noel,lefty,fitz,craig,anonymous,jluis
   ```

2. Haga que la propiedad y permisos del repositorio reflejen este nuevo grupo:

   ```
   floss$ cd /usr/local/nuevorepos
   floss$ chgrp -R cvs .
   floss$ chmod ug+rwx . CVSROOT
   ```

Ahora cualquiera de los usuarios listados en el grupo podrá empezar un proyecto ejecutando `cvs import` como se describió en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩. Las órdenes "checkout", "update" y "commit" también deberían funcionar.

También podrán entrar en el repositorio desde localizaciones remotas usando el método `:ext:`, asumiendo que tienen acceso por rsh o ssh a la máquina del repositorio. (Se habrá percatado de que las órdenes "chgrp" y "chmod" en el ejemplo de arriba le dieron acceso de escritura a un usuario llamado `anonymous`, que no es lo que uno esperaría. La razón es que incluso los usuarios anónimos y de sólo lectura del repositorio necesitan acceso de escritura a nivel del sistema, para que sus procesos CVS puedan crear ficheros de bloqueo temporales dentro del repositorio. CVS no asegura la restricción de "sólo lectura" del acceso anónimo por medio de permisos de ficheros Unix sino por otros medios, de lo que se hablará en ⟨undefined⟩ [Acceso anonimo], page ⟨undefined⟩.)

Si su repositorio está destinado a servir proyectos al público en general, en cuyo caso los contribuidores no tendrán necesariamente cuentas en la máquina del repositorio, debería configurar ahora el servidor de autentificación de contraseñas (see ⟨undefined⟩ [El servidor de autentificacion de contrasen˜as], page ⟨undefined⟩). Es necesario para acceso anónimo de sólo lectura, y seguramente sea la manera más fácil de asegurar acceso al envío de cambios a ciertas personas sin tener que darles cuentas completas en la máquina.

El servidor de autentificacion de contrasen˜as

Antes de seguir los pasos necesarios para configurar el servidor de contraseñas vamos a examinar cómo funcionan este tipo de conexiones en teoría. Cuando un cliente remoto CVS usa el método `:pserver:` para conectarse a un repositorio, el cliente está contactando en realidad con un número de puerto específico en la máquina servidora – en concreto el número de puerto 2401 (que es 49 al cuadrado, si le interesan este tipo de cosas). El puerto 2401 es el puerto designado por defecto para el servidor pserver de CVS, aunque se podría configurar para usar un puerto diferente siempre que el cliente y el servidor estén de acuerdo en ello.

El servidor CVS en realidad no está esperando conexiones a ese puerto – el servidor no empezará hasta que realmente llegue una conexión. En vez de ello, el programa Unix "inetd" (InterNET Daemon) está escuchando en ese puerto, y necesita saber que cuando reciba una petición de conexión ahi, debería iniciar el servidor CVS y conectarlo al cliente entrante.

Esto se consigue modificando los ficheros de configuración de inetd: '`/etc/services`' y '`/etc/inetd.conf`'. El fichero de servicios asigna números de puerto a nombres de servicios e inetd.conf le dice a inetd qué hacer para un nombre de servicio dado.

Primero ponga una línea como ésta en /etc/services (después de asegurarse de que la línea no existe ya):

```
cvspserver 2401/tcp
```

Luego, escriba esto en /etc/inetd.conf:

```
cvspserver stream tcp nowait root /usr/local/bin/cvs cvs \
   --allow-root=/usr/local/nuevorepos pserver
```

(En el fichero real, esto deberá ser una única línea larga, sin barra inversa \). Si su sistema usa una envoltura de TCP ("tcp wrapper", N. del T.), puede que quiera usar algo como esto en vez de lo anterior:

```
cvspserver stream tcp nowait root /usr/sbin/tcpd /usr/local/bin/cvs \
   --allow-root=/usr/local/nuevorepos pserver
```

Ahora reinicie inetd para que tenga en cuenta los cambios en sus ficheros de configuración (si no sabe cómo reiniciar el demonio, simplemente reinicie la máquina – esto también funcionará).

Esto es suficiente para permitir conexiones, pero también querrá configurar contraseñas especiales de CVS – separadas de las contraseñas de login de los usuarios – de modo que la gente pueda acceder al repositorio sin poner en peligro la seguridad general del sistema.

El fichero de contraseñas de CVS es CVSROOT/passwd en el repositorio. No se creó por defecto cuando ejecutó cvs init, porque CVS no sabe seguro si usará pserver. Incluso si el fichero de contraseñas se ha creado, CVS no tendrá forma de saber los nombres de usuario y contraseñas a crear. Así que usted tendrá que crear uno por sí mismo; aquí hay una muestra de fichero CVSROOT/passwd:

```
kfogel:rKa5jzULzmh0o
anonymous:XR4EZcEs0szik
melissa:tGX1fS8sun6rY:pubcvs
```

El formato es tan simple como parece. Cada línea es:

```
<NOMBREUSUARIO>:<CONTRASEÑA_CIFRADA>:<NOMBREUSUARIO_SISTEMA_OPCIONAL>
```

Los dos puntos adicionales seguidos de un nombre de usuario de sistema opcional le dicen a CVS que las conexiones autentificadas con NOMBREUSUARIO deberían ejecutarse como la cuenta de sistema NOMBREUSUARIO_SISTEMA – en otras palabras, que la sesión CVS sólo sería capaz de hacer en el repositorio las cosas que alguien conectado como NOMBREUSUARIO_SISTEMA podría hacer.

Si no se da un nombre de usuario de sistema, NOMBREUSUARIO deberá coincidir con un nombre de cuenta real del sistema, y la sesión se ejecutará con los permisos de ese usuario. En cualquier caso, la contraseña cifrada no debería ser la misma que la contraseña real de acceso del usuario. Debería ser una contraseña independiente usada sólo para conexiones a CVS pserver.

La contraseña se cifra usando el mismo algoritmo que las contraseñas estándar de Unix, almacenadas en /etc/passwd. Puede que se pregunte en este punto, cómo se consigue una versión cifrada de una contraseña? Para las contraseñas de sistema Unix, la orden passwd se encarga del cifrado en /etc/passwd por usted. Por desgracia no hay una orden equivalente a passwd en cvs (se ha propuesto varias veces, pero nadie se ha puesto a escribirlo – lo hará usted. quizá?).

Esto es un inconveniente, pero sólo pequeño. Si no hay otra opción, siempre podrá cambiar temporalmente la contraseña de sistema de un usuario usando passwd, copiar y pegar el texto cifrado de /etc/passwd en CVSROOT/passwd, y restaurar la antigua contraseña (en ciertos sistemas las contraseñas cifradas se encuentran en /etc/shadow y sólo el administrador o root puede leerlas.)

Este proceso es factible pero bastante incómodo. Sería mucho más fácil tener una utilidad de línea de órdenes que tomara una contraseña en texto plano como su argumento y diera como salida la versión cifrada. Aquí está esa herramienta, escrita en Perl:

```
#!/usr/bin/perl

srand (time());
my $randletter = "(int (rand (26)) + (int (rand (1) + .5) % 2 ? 65 : 97))";
my $salt = sprintf ("%c%c", eval $randletter, eval $randletter);
```

```
my $plaintext = shift;
my $crypttext = crypt ($plaintext, $salt);

print "${crypttext}\n";
```

Yo guardo el guión anterior en '`/usr/local/bin/cryptout.pl`':

```
floss$ ls -l /usr/local/bin/cryptout.pl

-rwxr-xr-x   1   root   root   265  Jun 14 20:41 /usr/local/bin/cryptout.pl
floss$ cryptout.pl "some text"
sB3A79YDX5L4s

floss$
```

Si usara la salida de este ejemplo para crear la siguiente entrada en CVSROOT/passwd

```
jluis:sB3A79YDX5L4s:craig
```

entonces la gente podría conectarse al repositorio con la siguiente orden:

```
remote$ cvs -d :pserver:jluis@floss.red-bean.com:/usr/local/nuevorepos login
```

Escribirían entonces `some text` como contraseña y a partir de entonces podrían ejecutar órdenes CVS con los mismos privilegios de acceso que el usuario de sistema `craig`.

Si alguien intenta autentificarse con un nombre de usuario y contraseña que no aparecen en CVSROOT/passwd, CVS comprobará si ese nombre de usuario y contraseña están presentes en /etc/passwd. Si lo están (y si la contraseña coincide, por supuesto), CVS proporcionará el acceso. Se comporta de esta forma para comodidad del administrador, para no tener que añadir entradas a CVSROOT/passwd por separado para los usuarios comunes del sistema. Sin embargo, este comportamiento también es un agujero de seguridad, porque significa que si uno de esos usuarios se conecta al servidor CVS, su contraseña de acceso al sistema circulará por la red en texto claro, potencialmente vulnerable a los ojos de husmeadores de contraseñas. Un poco más adelante, aprenderá cómo desactivar este comportamiento "problemático", para que CVS consulte sólo su propio fichero passwd. Tanto si lo deja activado o desactivado, probablemente debería obligar a los usuarios de CVS que también tengan cuentas en el sistema a mantener contraseñas distintas para las dos funciones.

Aunque el fichero passwd autentifica para todo el repositorio, con un poco de trabajo adicional podrá usarlo incluso para proporcionar acceso específico de proyecto. Aquí hay un método:

Suponga que quiere proporcionar acceso a algunos desarrolladores remotos al proyecto `foo`, y a otros acceso al proyecto `bar`, y no quiere que los desarrolladores de un proyecto tengan acceso al envío de cambios al otro. Puede conseguir esto creando cuentas de usuario y grupos específicos de proyecto en el sistema y luego referirse a esas cuentas en el fichero CVSROOT/passwd.

Aquí está el extracto relevante de /etc/passwd en cuestión

```
cvs-foo:*:600:600:Cuenta Pública CVS para el Proyecto Foo:/usr/local/cvs:/bin/false
cvs-bar:*:601:601:Cuenta Pública CVS para el Proyecto Bar:/usr/local/cvs:/bin/false
```

y de /etc/group

```
cvs-foo:*:600:cvs-foo
```

```
cvs-bar:*:601:cvs-bar
```

y, finalmente, CVSROOT/passwd:

```
kcunderh:rKa5jzULzmhOo:cvs-foo
jmankoff:tGX1fS8sun6rY:cvs-foo
brebard:cAXVPNZN6uFH2:cvs-foo
xwang:qp5lsf7nzRzfs:cvs-foo
dstone:JDNNF6HeX/yLw:cvs-bar
twp:glUHEM8KhcbO6:cvs-bar
ffranklin:cG6/6yXbS9BHI:cvs-bar
yyang:YoEqcCeCUq1vQ:cvs-bar
```

Algunos de los nombres de usuario de CVS se refieren a las cuentas de usuario de sistema `cvs-foo` y otras a `cvs-bar`. Dado que CVS se ejecuta bajo la ID de usuario de la cuenta de sistema, simplemente tendrá que asegurarse de que en las partes de interés del repositorio sólo pueden escribir los usuarios y grupos adecuados. Si se asegura de que las cuentas de usuario de sistema estén bien atadas (sin contraseña de acceso al sistema válida, con '`/bin/false`' como shell), el sistema será razonablemente seguro (pero mire más adelante en este capítulo acerca de los permisos CVSROOT!). Además, CVS registra los cambios e informes de cambios bajo el nombre de usuario de CVS, no bajo el nombre de usuario de sistema, así que usted podrá saber quién es responsable de un cambio dado.

Acceso anonimo

Hasta ahora sólo hemos visto cómo usar el servidor de autentificación de contraseñas para dar acceso total al repositorio (aunque es cierto que se puede restringir ese acceso mediante permisos de fichero Unix cuidadosamente elegidos). Pasar a acceso anónimo y de sólo lectura es un paso simple: sólo hay que añadir un nuevo fichero, o quizá dos, en CVSROOT/. Los nombres de los ficheros son `readers` y `writers` – el primero contiene una lista de nombres de usuario que pueden leer el repositorio solamente, y el segundo los usuarios que pueden leer y escribir.

Si lista un nombre de usuario en CVSROOT/readers, ese usuario tendrá acceso de sólo lectura a todos los proyectos del repositorio. Si lista un nombre de usuario en CVSROOT/writers, ese usuario tendrá acceso a escritura, y todos los usuarios de pserver que no estén listados en writers tendrán acceso de sólo lectura (es decir, si el fichero writers existe, implica acceso de sólo lectura para todos los que no estén listados en él). Si el mismo nombre de usuario se encuentra listado en los dos ficheros, CVS resuelve el conflicto del modo más conservador: el usuario tendrá acceso de sólo lectura.

El formato de los ficheros es muy simple: un usuario por línea (no olvide poner una nueva línea en blanco después del último usuario). Un fichero readers de muestra sería:

```
anonymous
splotnik
guest
jbrowse
```

Hay que notar que los ficheros se refieren a nombres de usuario de CVS, no de sistema. Si usa alias de usuario en el fichero CVSROOT/passwd (poniendo un nombre de usuario de sistema después de los segundos dos puntos), el nombre de usuario más a la izquierda es el que hay que listar en el fichero readers o writers.

Para ser preciso, hay una descripción formal del comportamiento del servidor para decidir si dar acceso de sólo lectura o de lectura y escritura:

Si existe un fichero readers y este usuario está listado en él, se le dará acceso de sólo lectura. Si existe un fichero writers y este usuario no está listado en él, se le dará también acceso de sólo lectura (esto es cierto incluso si existe un fichero readers pero esa persona no está listada en él). Si esa persona está listada en ambos, se le dará acceso de sólo lectura. En todos los demás casos, a esa persona se le dará acceso completo de lectura y escritura.

Así, un repositorio típico con acceso a CVS anónimo tendrá esto (o algo parecido) en CVSROOT/passwd

```
anonymous:XR4EZcEs0szik
```

esto (o algo parecido) en /etc/passwd

```
anonymous:!:1729:105:Usuario CVS Anónimo:/usr/local/nuevorepos:/bin/false
```

y esto en CVSROOT/readers:

```
anonymous
```

Y, por supuesto, la configuración mencionada anteriormente en /etc/services y /etc/inetd.conf. Y eso es todo!

Hay que remarcar que algunos sistemas Unix antiguos no permiten nombres de usuario mayores de ocho caracteres. Una forma de arreglar esto sería llamar al usuario `anon` en lugar de `anonymous` en CVSROOT/passwd y los ficheros de sistema, porque la gente asume a menudo que anon es una abreviatura de anonymous de todas formas. Pero sería mejor poner algo como esto en el fichero CVSROOT/passwd

```
anonymous:XR4EZcEs0szik:cvsanon
```

(y luego, por supuesto, usar `cvsanon` en los ficheros de sistema). De este modo será capaz de publicar una dirección de repositorio que use `anonymous`, que es más o menos el estándar ahora. La gente que acceda al repositorio con

```
cvs -d :pserver:anonymous@cvs.foobar.com:/usr/local/nuevorepos (etc...)
```

realmente ejecutarían en el servidor como cvsanon (o lo que sea). Pero no necesitarían conocer o preocuparse de cómo están configuradas las cosas en el lado del servidor – sólo verían la dirección publicada.

Estructura del repositorio

El nuevo repositorio aún no tiene proyectos en él. Vamos a volver a ejecutar la importación inicial de ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩, observando lo que le ocurre al repositorio. (Por simplicidad, todos las órdenes asumen que la variable de entorno CVSROOT tiene el valor /usr/local/nuevorepos, así que no hay necesidad de especificar el repositorio con -d en importaciones y comprobaciones.)

```
floss$ ls /usr/local/nuevorepos
CVSROOT/
floss$ pwd
/home/jluis/src/
floss$ ls
miproyecto/
floss$ cd miproyecto
```

```
floss$ cvs import -m "importación inicial a CVS" miproyecto jluis start
N miproyecto/README.txt
N miproyecto/hello.c
cvs import: Importing /usr/local/nuevorepos/miproyecto/a-subdir
N miproyecto/a-subdir/loquesea.c
cvs import: Importing /usr/local/nuevorepos/miproyecto/a-subdir/subsubdir
N miproyecto/a-subdir/subsubdir/fish.c
cvs import: Importing /usr/local/nuevorepos/miproyecto/b-subdir
N miproyecto/b-subdir/random.c

No conflicts created by this import

floss$ ls /usr/local/nuevorepos
CVSROOT/  miproyecto/
floss$ cd /usr/local/nuevorepos/miproyecto
floss$ ls
README.txt,v  a-subdir/     b-subdir/   hello.c,v
floss$ cd a-subdir
floss$ ls
subsubdir/    loquesea.c,v
floss$ cd ..

floss$
```

Antes de importar, el repositorio contenía sólo su área de administración, CVSROOT. Después de la importación ha aparecido un nuevo directorio – '`miproyecto`' –. Los ficheros y subdirectorios existentes en ese nuevo directorio se parecen sospechosamente a los del proyecto que hemos importado, excepto que los ficheros tienen el sufijo `,v`. Éstos son ficheros de control de versión en formato RCS (la `,v` quiere decir "versión"), y son el esqueleto del repositorio. Cada fichero RCS almacena la historia de revisiones de su correspondiente fichero del proyecto, incluyendo todas las ramas y marcas.

Formato RCS

No necesita conocer nada del formato RCS para usar CVS (aunque hay un escrito excelente incluido en la distribución fuente, vea doc/RCSFILES). Sin embargo, una comprensión básica del formato puede ser de inmensa ayuda para resolver problemas con CVS, así que echaremos un pequeño vistazo a uno de los ficheros, '`hello.c,v`'. Aquí está su contenido:

```
head     1.1;
branch   1.1.1;
access   ;
symbols  start:1.1.1.1 jluis:1.1.1;
locks    ; strict;
comment  @ * @;

1.1
date     99.06.20.17.47.26;  author jluis;  state Exp;
branches 1.1.1.1;
next;
```

```
1.1.1.1
date     99.06.20.17.47.26;  author jluis;  state Exp;
branches ;
next;

desc
@@

1.1
log
@Initial revision
@
text
@#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
}
@

1.1.1.1
log
@importación inicial a CVS
@
text
@@
```

Uff! La mayoría de esto se puede ignorar; no hace falta que se preocupe de la relación entre 1.1 y 1.1.1.1, por ejemplo, o de la rama implicada 1.1.1 – en realidad no son significativas, desde un punto de vista del usuario o incluso del administrador. Lo que debería comprender es el formato en general. Al comienzo hay una colección de cabeceras:

```
head     1.1;
branch   1.1.1;
access   ;
symbols  start:1.1.1.1 jluis:1.1.1;
locks    ; strict;
comment  @ * @;
```

Más abajo hay grupos de metainformación sobre cada revisión (pero aún sin mostrar el contenido de esa revisión), como:

```
1.1
date     99.06.20.17.47.26;  author jluis;  state Exp;
branches 1.1.1.1;
next     ;
```

Y finalmente, el informe de cambios ("log message", N. del T.) y texto de una revisión real:

```
1.1
```

```
log
@Initial revision
@
text
@#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
}
@

1.1.1.1
log
@importación inicial a CVS
@
text
@@
```

Si lo mira de cerca verá que el contenido de la primera revisión se guarda bajo la cabecera 1.1, pero en ella el informe de cambios es "Initial revision", mientras que el mensaje que usamos en realidad a la hora de importar fue "importación inicial a CVS". No es necesario que se preocupe por esta discrepancia ahora. Ocurre porque las importaciones son circunstancias especiales: para que importaciones repetidas en el mismo proyecto tengan un efecto útil, la importación en realidad coloca la revisión inicial en el tronco principal y en una rama especial (las razones para ello se aclararán cuando veamos derivaciones comerciales en ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩). Por ahora puede tratar `1.1` y `1.1.1.1` como la misma cosa.

El fichero se vuelve aún más revelador después de que enviemos con commit la primera modificación a hello.c:

```
floss$ cvs -Q co miproyecto
floss$ cd miproyecto
floss$ emacs hello.c
    (haga algunos cambios al fichero)

floss$ cvs ci -m "ahora también dice adiós"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in hello.c;
/usr/local/nuevorepos/miproyecto/hello.c,v  <--  hello.c
new revision: 1.2; previous revision: 1.1
done
```

Si mira en el repositorio a hello.c,v verá el efecto del envío de cambios:

```
head  1.2;
access;
symbols
```

```
      start:1.1.1.1 jluis:1.1.1;
locks; strict;
comment   @ * @;

1.2
date   99.06.21.01.49.40;   author jluis;   state Exp;
branches;
next   1.1;

1.1
date   99.06.20.17.47.26;   author jluis;   state Exp;
branches
       1.1.1.1;
next   ;

1.1.1.1
date   99.06.20.17.47.26;   author jluis;   state Exp;
branches;
next   ;

desc
@@

1.2
log
@ahora también dice adiós
@
text
@#include <stdio.h>

void
main ()
{
  printf ("Hola, mundo!\n");
  printf ("Adiós, mundo!\n");
}
@

1.1
log
@Initial revision
@
text
@d7 1
@

1.1.1.1
log
@importación inicial a CVS
```

```
@
text
@@
```

Ahora el contenido completo de la revisión 1.2 está guardado en el fichero, y el texto para la revisión 1.1 ha sido reemplazado por la fórmula críptica:

```
d7 1
```

El `d7 1` es un código diff que quiere decir "empezando en la línea 7, borrar 1 línea". En otras palabras, para obtener la Revisión 1.1, borre la línea 7 de la Revisión 1.2! Pruébelo usted mismo. Verá que de hecho produce la Revisión 1.1 – simplemente se deshace de la línea que añadimos al fichero.

Esto demuestra el principio básico del formato RCS: Almacena sólo las diferencias entre revisiones, ahorrando con ello un montón de espacio comparado con guardar cada revisión entera. Para volver desde la última revisión a la anterior, parchea la última revisión usando el diff almacenado. Por supuesto, esto significa que cuanto más hacia atrás viaje en la historia de revisiones, habrá que realizar más operaciones de parcheo (por ejemplo, si el fichero está en la Revisión 1.7 y a CVS se le pide que muestre la Revisión 1.4, tendrá que producir la 1.6 parcheando hacia atrás la 1.7, luego la 1.5 parcheando la 1.6, y finalmente la 1.4 parcheando la 1.5). Por suerte, las revisiones antiguas son además las menos solicitadas, así que el sistema RCS funciona bastante bien en la práctica: Cuanto más reciente sea una revisión, más "barata" es de obtener.

En cuanto a la información de cabecera al principio del fichero, no necesita saber lo que significa todo ello. Sin embargo, los efectos de ciertas operaciones se muestran muy claramente en las cabeceras, y una pequeña familiaridad con ellas puede resultar útil.

Cuando envía cambios de una nueva revisión al tronco, la etiqueta `head` se actualiza (note cómo cambió a 1.2 en el ejemplo anterior, cuando se envió el cambio de la segunda revisión a hello.c). Cuando añade un fichero como binario o lo marca, esas operaciones se registran también en las cabeceras. Como ejemplo, vamos a añadir foo.jpg como fichero binario para después marcarlo un par de veces:

```
floss$ cvs add -kb foo.jpg
cvs add: scheduling file 'foo.jpg' for addition
cvs add: use 'cvs commit' to add this file permanently
floss$ cvs -q commit -m "añadida una imagen aleatoria; pregunte a \
   jluis@red-bean.com el motivo"
RCS file: /usr/local/nuevorepos/miproyecto/foo.jpg,v
done
Checking in foo.jpg;
/usr/local/nuevorepos/miproyecto/foo.jpg,v  <--  foo.jpg
initial revision: 1.1
done
floss$ cvs tag alguna_marca_aleatoria foo.jpg
T foo.jpg
floss$ cvs tag OTRA-MARCA foo.jpg
T foo.jpg
floss$
```

Examine ahora la sección "header" de foo.jpg,v en el repositorio:

```
head   1.1;
```

```
access;
symbols
      OTRA-MARCA:1.1
      alguna_marca_aleatoria:1.1;
locks; strict;
comment   @# @;
expand @b@;
```

Fíjese en la b en la línea "expand" del final – se debe a haber usado el parámetro -kb al añadir el fichero, y quiere decir que el fichero no sufrirá expansiones de palabra clave o nueva línea, que ocurrirían normalmente durante obtenciones de copia y actualizaciones si fuera un fichero de texto normal. Las marcas aparecen en la sección "symbols", una por línea – ambas están asociadas a la primera revisión, puesto que eso es lo que se marcó ambas veces. (Esto también ayuda a explicar por qué los nombres de marca pueden sólo contener letras, números, guiones y guiones bajos. Si la propia marca contuviera puntos o comas, su registro RCS podría ser ambiguo, porque no habría forma de encontrar el enlace textual entre la marca y la revisión a la que está asociada.)

El formato RCS siempre va entre signos @

El símbolo @ se usa como delimitador de campos en los ficheros RCS, lo que significa que si aparece alguno en el texto de un fichero o en un informe de cambios, deberá estar comentado (de lo contrario, CVS interpretaría incorrectamente que está marcando el final de ese campo). Se comenta poniéndolo doble – es decir, CVS siempre interpreta @@ como un "signo @ literal", nunca como un "fin de campo actual". Cuando enviamos los cambios a foo.jpg, el informe de cambios fue

```
"añadida una imagen aleatoria; pregunte a jluis@red-bean.com el motivo"
```

que se almacena en foo.jpg,v así:

```
1.1
log
@añadida una imagen aleatoria; pregunte a jluis@@red-bean.com el motivo
@
```

El signo @ en jluis@@red-bean.com se descomentará automáticamente cada vez que CVS obtenga el informe de cambios:

```
floss$ cvs log foo.jpg
RCS file: /usr/local/nuevorepos/miproyecto/foo.jpg,v
Working file: foo.jpg
head: 1.1
branch:
locks: strict
access list:
symbolic names:
      OTRA-MARCA: 1.1
      alguna_marca_aleatoria: 1.1
keyword substitution: b
total revisions: 1; selected revisions: 1
description:
----------------------------
```

```
revision 1.1
date: 1999/06/21 02:56:18;  author: jluis;  state: Exp;
añadida una imagen aleatoria: pregunte a jluis@red-bean.com el motivo
=============================================================================

floss$
```

El único motivo por el que debería preocuparse es por si alguna vez tiene que editar a mano ficheros RCS (una circunstancia rara, aunque le ha pasado a más de uno) Debe acordarse entonces de usar signos dobles `@` en contenidos de la revisión e informes de cambios. Si no lo hace, el fichero RCS estará corrupto y probablemente tendrá un comportamiento extraño e indeseable.

Hablando de editar a mano ficheros RCS, no se deje engañar por los permisos en el repositorio:

```
floss$ ls -l
total 6
-r--r--r--   1 jluis   users         410 Jun 20 12:47 README.txt,v
drwxrwxr-x   3 jluis   users        1024 Jun 20 21:56 a-subdir/
drwxrwxr-x   2 jluis   users        1024 Jun 20 21:56 b-subdir/
-r--r--r--   1 jluis   users         937 Jun 20 21:56 foo.jpg,v
-r--r--r--   1 jluis   users         564 Jun 20 21:11 hello.c,v

floss$
```

(Para los que no estén familiarizados con la salida de "ls" en Unix, las líneas `-r--r--r--` de la izquierda básicamente quieren decir que los ficheros se pueden leer pero no cambiar.) Aunque los ficheros parecen ser de sólo lectura para todos, también hay que tener en cuenta los permisos de directorio:

```
floss$ ls -ld .
drwxrwxr-x   4 jluis   users        1024 Jun 20 22:16 ./
floss$
```

El propio directorio miproyecto/ – y sus subdirectorios – es accesible para escritura por el propietario (jluis) y el grupo (users). Esto significa que CVS (ejecutándose como jluis o como cualquiera del grupo users) puede crear y borrar ficheros en esos directorios, incluso si no puede editar directamente los ficheros a presentes. CVS edita un fichero RCS haciendo una copia separada de él, de forma que usted haga todos sus cambios en una copia temporal, y luego reemplaza el fichero RCS existente con el nuevo. (Pero por favor, no pregunte por qué los ficheros son de sólo lectura – hay razones históricas para ello, relacionadas con la forma en que RCS trabaja cuando se ejecuta como programa en solitario.)

Por cierto, puede que usted no desee que el grupo de los ficheros sea `users`, considerando que el directorio raíz del repositorio se le asignó explícitamente el grupo `cvs`. Puede corregir el problema ejecutando esta orden dentro del repositorio:

```
floss$ cd /usr/local/nuevorepos
floss$ chgrp -R cvs miproyecto
```

Las reglas habituales Unix de creación de ficheros rigen qué grupo se asigna a los nuevos ficheros que aparecen en el repositorio, así que de vez en cuando puede que necesite ejecutar "chgrp" o "chmod" en ciertos ficheros o directorios del repositorio (ajustar el bit SGID con `chmod g+s` es a menudo una buena estrategia: hace que los hijos de un directorio hereden

el grupo propietario del directorio, que por lo general es lo que quiere que pase en el repositorio). No hay reglas rápidas acerca de cómo debería estructurar los permisos del repositorio; depende de quién esté trabajando en qué proyectos.

Qué ocurre cuando elimina un fichero

Cuando elimina un fichero de un proyecto, no desaparece simplemente. CVS debe ser capaz de recuperar esos ficheros cuando solicite una revisión antigua del proyecto. En lugar de ello, el fichero se pone en el `Attic`, ático literalmente:

```
floss$ pwd
/home/jluis/src/miproyecto
floss$ ls /usr/local/nuevorepos/miproyecto/
README.txt,v  a-subdir/     b-subdir/     foo.jpg,v   hello.c,v
floss$ rm foo.jpg
floss$ cvs rm foo.jpg
cvs remove: scheduling 'foo.jpg' for removal
cvs remove: use 'cvs commit' to remove this file permanently
floss$ cvs ci -m "Eliminado foo.jpg" foo.jpg
Removing foo.jpg;
/usr/local/nuevorepos/miproyecto/foo.jpg,v  <--  foo.jpg
new revision: delete; previous revision: 1.1
done
floss$ cd /usr/local/nuevorepos/miproyecto/
floss$ ls
Attic/      README.txt,v  a-subdir/     b-subdir/   hello.c,v
floss$ cd Attic
floss$ ls
foo.jpg,v
floss$
```

En cada directorio del repositorio de un proyecto, la presencia de un subdirectorio '`Attic/`' indica que se ha borrado al menos un fichero de ese directorio (esto quiere decir que no debería usar directorios llamados Attic en sus proyectos). Sin embargo, CVS no mueve simplemente el fichero RCS a Attic/; además envía el cambio con una nueva revisión al fichero, con un estado especial de revisión de `dead`, muerto. Aquí está la sección de interés de Attic/foo.jpg,v:

```
1.2
date   99.06.21.03.38.07;   author jluis;   state dead;
branches;
next 1.1;
```

Si el fichero se vuelve a traer de nuevo a la vida, CVS tiene una forma de registrar que estaba muerto en algún punto del pasado y que ahora está vivo otra vez.

Esto quiere decir que si quiere restaurar un fichero eliminado, no puede sacarlo del Attic/ simplemente y ponerlo de nuevo en el proyecto. En lugar de ello, tiene que hacer algo como lo siguiente con una copia de trabajo:

```
floss$ pwd
/home/jluis/src/miproyecto
floss$ cvs -Q update -p -r 1.1 foo.jpg > foo.jpg
```

```
floss$ ls
CVS/       README.txt   a-subdir/   b-subdir/   foo.jpg     hello.c
floss$ cvs add -kb foo.jpg
cvs add: re-adding file foo.jpg (in place of dead revision 1.2)
cvs add: use 'cvs commit' to add this file permanently
floss$ cvs ci -m "revivida imagen jpg" foo.jpg
Checking in foo.jpg;
/usr/local/nuevorepos/miproyecto/foo.jpg,v  <-- foo.jpg
new revision: 1.3; previous revision: 1.2
done
floss$ cd /usr/local/nuevorepos/miproyecto/
floss$ ls
Attic/       a-subdir/     foo.jpg,v
README.txt,v  b-subdir/     hello.c,v
floss$ ls Attic/
floss$
```

Queda mucho más por saber del formato RCS, pero esto es suficiente para que un administrador de CVS mantenga un repositorio. Es bastante raro que realmente haya que editar un fichero RCS; normalmente sólo tendrá que ajustar permisos de ficheros en el repositorio, al menos si mi propia experiencia sirve de guía. Sin embargo, cundo CVS empieze a comportarse de forma realmente extraña (raro, pero no completamente fuera de lo posible), puede que quiera mirar dentro de los ficheros RCS para averiguar qué está pasando.

El directorio administrativo CVSROOT/

Los ficheros de nuevorepos/CVSROOT/ no son parte de ningún proyecto, sino que se usan para controlar el comportamiento de CVS en el repositorio. La mejor forma de editar esos ficheros es obtener una copia de trabajo de CVSROOT con "checkout", igual que para un proyecto normal:

```
floss$ cvs co CVSROOT
cvs checkout: Updating CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
U CVSROOT/cvswrappers
U CVSROOT/editinfo
U CVSROOT/loginfo
U CVSROOT/modules
U CVSROOT/notify
U CVSROOT/rcsinfo
U CVSROOT/taginfo
U CVSROOT/verifymsg
floss$
```

Miraremos los ficheros en order aproximado de importancia. Fíjese en que cada uno de los ficheros viene con un comentario explicativo al comienzo (el convenio para comentarios es el mismo para todos ellos: un signo # al principio de la línea indica un comentario, y CVS ignora estas líneas cuando procesa los ficheros). Recuerde que cualquier cambio que haga a

los ficheros de administración de la copia de trabajo obtenida no afectará al comportamiento de CVS hasta que envíe los cambios.

Si usted es extremadamente consciente de la seguridad, puede que quiera configurar los permisos de fichero en CVSROOT para que sean diferentes de los permisos en cualquier parte del repositorio, para tener bien controlado quién puede enviar cambios a CVSROOT. Como verá un poco más adelante, el ser capaz de modificar los ficheros de CVSROOT básicamente le da a cualquier usuario CVS – incluso a los remotos – la capacidad de ejecutar cualquier orden en la máquina del repositorio.

El Fichero config

El fichero *config* le permite configurar ciertos parámetros de comportamiento global. Sigue un formato muy estricto

```
PARÁMETRO=VALOR
(etc)
```

sin permitirse espacios adicionales. Por ejemplo, aquí hay un posible fichero config:

```
SystemAuth=yes
TopLevelAdmin=no
PreservePermissions=no
```

(Una entrada ausente equivaldría a `no`.)

El parámetro `SystemAuth` controla si CVS debería mirar en el fichero de sistema passwd si falla al encontrar un determinado nombre de usuario en el fichero CVSROOT/passwd. Las distribuciones de CVS vienen con él puesto a `no` para ser conservadoras en cuanto a la seguridad del sistema.

`TopLevelAdmin` le dice a CVS si hacer un directorio CVS/ hermano cuando obtiene una copia de trabajo. Este directorio CVS/ podría no estar dentro de la copia de trabajo, sino junto a ella. Sería conveniente habilitarlo si usted tiende (y los usuarios del repositorio tienden) a obtener copias de muchos proyectos diferentes en el mismo repositorio. De lo contrario debería dejarlo desactivado, ya que puede ser desconcertante ver que aparece un directorio CVS/ adicional donde no lo espera.

`PreservePermissions` controla si se conservan los permisos de fichero y otra metainformación en la historia de revisiones. Ésta es una característica algo oscura que seguramente no valga la pena describir en detalle. Si está interesado vea el nodo *Special Files* en el Cederqvist (*nodo* es una palabra de Texinfo para una posición particular en un documento Info. Para ir a un nodo mientras se lee Info, teclee *g* seguido del nombre del nodo, desde cualquier parte del documento).

`LockDir` también es una característica usada raramente. En circunstancias especiales, querrá decirle a CVS que cree sus ficheros de bloqueo en algún sitio distinto de los subdirectorios del proyecto, para evitar problemas de permisos. Estos ficheros de bloqueo impiden que CVS tropiece consigo mismo al realizar múltiples operaciones en el mismo directorio del repositorio a la vez. En general, nunca tendrá que preocuparse por ello, pero a veces puede que los usuarios tengan problemas actualizando u obteniendo copias de trabajo desde un directorio del repositorio porque son incapaces de crear un fichero de bloqueo (CVS necesita crear un fichero de bloqueo, incluso en operaciones de sólo lectura, para evitar situaciones en las que podría acabar leyendo mientras otra invocación de CVS está escribiendo). El

remedio habitual para esto es cambiar los permisos del repositorio, pero cuando esto no es factible, el parámetro LockDir puede resultar práctico.

No hay más parámetros por el momento, pero puede que en versiones futuras de CVS se añadan otros nuevos; debería comprobar siempre el Cederqvist o el propio fichero config de la distribución para ver los cambios.

El Fichero modules

En modules puede definir aliases y grupos alternativos de proyectos en el repositorio. La línea de module más básica es de la forma:

```
NOMBRE_MÓDULO   DIRECTORIO_EN_REPOSITORIO
```

por ejemplo,

```
mp    miproyecto
asub  miproyecto/a-subdir
```

(Las rutas dadas a la derecha son relativas al directorio raíz del repositorio.) Esto les da a los desarrolladores un nombre alternativo con el que obtener una copia de un proyecto o parte de un proyecto:

```
floss$ cvs co mp
cvs checkout: Updating mp
U mp/README.txt
U mp/foo.jpg
U mp/hello.c
cvs checkout: Updating mp/a-subdir
U mp/a-subdir/loquesea.c
cvs checkout: Updating mp/a-subdir/subsubdir
U mp/a-subdir/subsubdir/fish.c
cvs checkout: Updating mp/b-subdir
U mp/b-subdir/random.c
```

o

```
floss$ cvs -d /usr/local/nuevorepos/ co asub
cvs checkout: Updating asub
U asub/loquesea.c
cvs checkout: Updating asub/subsubdir
U asub/subsubdir/fish.c
```

Note cómo en ambos casos el nombre del módulo pasó a ser el nombre del directorio creado para la copia de trabajo. En el caso de asub, ni siquiera se preocupó del directorio intermedio miproyecto/, sino que en lugar de ello creó un asub/ en un nivel superior, aunque venía de miproyecto/a-subdir en el repositorio. Las actualizaciones, envíos de cambios y todos las órdenes CVS funcionarán normalmente en esas copias de trabajo – lo único raro que tienen son sus nombres.

Poniendo nombres de ficheros después del nombre de directorio podrá definir un módulo consistente sólo en algunos de los ficheros de un directorio del repositorio dado. Por ejemplo

```
readme  miproyecto  README.txt
```

y

```
no-readme  miproyecto  hello.c  foo.jpg
```

permitirían las siguientes obtenciones de copia respectivamente:

```
floss$ cvs -q co readme
U readme/README.txt
floss$ cvs -q co no-readme
U no-readme/hello.c
U no-readme/foo.jpg
floss$
```

Puede definir un módulo que incluya múltiples directorios de repositorio usando la opción -a (para `alias`), pero fíjese en que se investigarán los directorios bajo sus nombres originales. Por ejemplo, esta línea

```
dosproyectos  -a  miproyecto  tuproyecto
```

le permitiría hacer esto (asumiendo que tanto miproyecto/ como tuproyecto/ están en el repositorio):

```
floss$ cvs co dosproyectos
U miproyecto/README.txt
U miproyecto/foo.jpg
U miproyecto/hello.c
U miproyecto/a-subdir/loquesea.c
U miproyecto/a-subdir/subsubdir/fish.c
U miproyecto/b-subdir/random.c
U tuproyecto/README
U tuproyecto/foo.c
U tuproyecto/un-subdir/fichero1.c
U tuproyecto/un-subdir/fichero2.c
U tuproyecto/un-subdir/otro-subdir/bla.c
```

El nombre `dosproyectos` es un recurso conveniente para meterse en los dos proyectos, pero no afecta a los nombres de las copias de trabajo. (Por cierto, no hay necesidad de que los módulos alias se refieran a múltiples directorios; podríamos haber omitido dosproyectos, en cuyo caso aún se habría obtenido una copia de miproyecto bajo el nombre `miproyecto`.)

Los módulos pueden incluso referirse a otros módulos, poniéndoles como prefijo un signo &:

```
mp    miproyecto
asub  miproyecto/a-subdir
dosproyectos -a miproyecto tuproyecto
dp  &dosproyectos
```

Hacer un checkout de `dp` tendría exactamente el mismo resultado que el de `dosproyectos`.

Hay algunos otros trucos que puede hacer con módulos, la mayoría de ellos más infrecuentes que los que se acaban de presentar. Vea el nodo modules en el Cederqvist para obtener información sobre ellos.

Los Ficheros commitinfo y loginfo y rcsinfo

La mayoría de los demás ficheros de administración proporcionan *puntos de comtrol* ("hooks", N. del T.) programáticos en varias partes del proceso de envío de cambios (por ejemplo, la capacidad de validar informes de cambios o estados de fichero antes de permitir el envío, o la capacidad de notificar a un grupo de desarrolladores cada vez que se haga un envío en cierto directorio del repositorio).

Por lo general, los ficheros comparten una sintaxis común. Cada línea es de la forma:

```
EXPRESIÓN_REGULAR    PROGRAMA_A_EJECUTAR
```

La expresión regular se probará con el directorio en el que se está haciendo el envío (con el nombre de directorio relativo al nivel más alto del repositorio). Si coincide se ejecutará el programa designado. Al programa se le pasarán los nombres de cada uno de los ficheros del envío; puede hacer lo que le parezca con esos nombres, incluso abrir los ficheros y examinar sus contenidos. Si el programa devuelve un estado de salida distinto de cero, se impide que se haga el envío.

Las (*expresiones regulares* son un sistema para describir de forma concisa clases de cadenas. Si no está familiarizado con las expresiones regulares, puede hacerlo con el siguiente resumen: `foo` coincidiría con todos los ficheros con nombres conteniendo `foo`; y `foo.*bar` coincidiría con todos los ficheros con nombres conteniendo `foo`, seguido de cualquier número de caracteres, y seguido por la cadena `bar`. Es por ello que las subcadenas normales coinciden consigo mismas, pero `.` y `*` son especiales. `.` coincide con cualquier caracter, y `*` quiere decir "coincide con cualquier número de veces, incluyendo cero, del caracter precedente". Los signos `^` y `$` indican "encuentra al principio y final de la cadena", respectivamente; por tanto, `^foo.*bar.*baz$` encontraría las cadenas que comenzaran con `foo`, que contuvieran `bar` en medio, y que acabaran con `baz`. Esto es todo lo que vamos a profundizar; este resumen es una pequeña parte muy abreviada de la sintaxis completa de las expresiones regulares.)

El fichero *commitinfo* está para puntos de control genéricos que quiera ejecutar en todos los envíos. Aquí hay algunas líneas commitinfo de ejemplo:

```
^a-subdir*      /usr/local/bin/comprobar-asubdir.sh
ou              /usr/local/bin/validar-proyecto.pl
```

Así que cualquier envío en miproyecto/a-subdir/ coincidiría con la primera línea, por lo que se ejecutaría el guión comprobar-asubdir.sh. Un envío en cualquier proyecto cuyo nombre (nombre real de directorio del repositorio, no necesariamente nombre del módulo) contenga la cadena `ou` ejecutaría el guión validar-proyecto.pl, a menos que el envío ya haya coincidido con la línea a-subdir anterior.

En lugar de una expresión regular se puede usar la palabra `DEFAULT` o `ALL`. La línea DEFAULT (o la primera línea DEFAULT si hay más de una) se ejecutará si no coincide ninguna expresión regular, y cada una de las líneas ALL se ejecutará además de todas las líneas que puedan coincidir.

Los nombres de ficheros que se pasan al programa no se refieren a ficheros RCS – hacen referencia a ficheros normales, cuyos contenidos son exactamente los mismos que los de la copia de trabajo de la que se están enviando cambios. El único aspecto inusual es que CVS los tiene guardados temporalmente dentro del repositorio, así que estarán disponibles para los programas que se estén ejecutando en la máquina en la que está el repositorio.

El fichero *loginfo* es similar a commitinfo, excepto en que en lugar de actuar en los contenidos de los ficheros, actúa en los informes de cambios. El lado izquierdo del fichero loginfo contiene expresiones regulares, quizá incluyendo líneas DEFAULT y ALL. El programa invocado a la derecha recibe el informe de cambios en su entrada estándar; puede hacer lo que quiera con esa entrada.

El programa de la derecha también puede admitir un número arbitrario de argumentos de línea de órdenes. Uno de esos argumentos puede ser un código especial `%`, a expandir por CVS en tiempo de ejecución como sigue:

```
%s    ------>     nombre(s) de fichero(s) afectados por el envío de cambios
%V    ------>     número(s) de revisión antes del envío de cambios
%v    ------>     número(s) de revisión después del envío de cambios
```

La expansión empieza siempre con el subdirectorio del repositorio (relativo al nivel superior del repositorio), seguido de la información del fichero. Por ejemplo, si los ficheros afectados por el envío de cambios fueran foo, bar y baz, todos en '`miproyecto/a-subdir`', `%s` se expandiría en:

```
miproyecto/a-subdir  foo  bar  baz
```

mientras que `%V` se expandiría para mostrar los números de revisión antiguos:

```
miproyecto/a-subdir  1.7  1.134  1.12
```

y `%v` los números de revisión nuevos:

```
miproyecto/a-subdir  1.8  1.135  1.13
```

Puede combinar expresiones con `%` delimitándolas con llaves siguiendo al signo `%` – esto las expandirá en series de sublistas separadas por comas, cada una conteniendo la información correspondiente a un fichero del envío. Por ejemplo, `%{sv}` se expandiría en

```
miproyecto/a-subdir  foo,1.8  bar,1.135  baz,1.13
```

y `%{sVv}` se expandiría en

```
miproyecto/a-subdir  foo,1.7,1.8  bar,1.134,1.135  baz,1.12,1.13
```

(Puede que tenga que mirar con cuidado para distinguir las comas de los puntos decimales en estos ejemplos.)

Aquí hay un fichero loginfo de ejemplo:

```
^miproyecto$   /usr/local/nuevorepos/CVSROOT/log.pl \
   -m miproyecto-devel@foobar.com %s
ou             /usr/local/bin/ou-notify.pl  %{sv}
DEFAULT        /usr/local/bin/default-notify.pl  %{sVv}
```

En la primera línea, cualquier envío de cambios en el subdirectorio miproyecto del repositorio invoca '`log.pl`', pasándole una dirección de correo electrónico (a la que '`log.pl`' enviará un correo con el informe de cambios), seguido del repositorio, seguido de todos los ficheros del envío.

En la segunda línea, cualquier envío de cambios en un subdirectorio del repositorio que contenga la cadena `ou` invocará el guión (imaginario) '`notificar-ou.pl`', pasándole el repositorio seguido de los nombres de los ficheros y de los nuevos números de revisión de los ficheros del envío.

La tercera línea invoca el guión (también imaginario) '`notificar-defecto.pl`' para cualquier envío que no coincida con ninguna de las dos líneas anteriores, pasándole toda la información posible (ruta al repositorio, nombres de fichero, revisiones antiguas y revisiones nuevas).

Los Ficheros verifymsg y rcsinfo

A veces puede que solamente quiera un programa que verifique que los informes de cambios se ajustan a un cierto estándar y que detenga el envío si no se cumple ese estándar. Esto puede conseguirse usando `verifymsg`, posiblemente con algo de ayuda de `rcsinfo`.

El fichero *verifymsg* es la combinación habitual de expresiones regulares y programas. El programa recibe el informe de cambios por la entrada estándar; es de suponer que realizará ciertas comprobaciones para verificar que el informe de cambios cumple ciertos criterios, y finalmente sale con estado cero o distinto de cero. En este último caso, el envío fallará.

Mientras tanto, el lado izquierdo de rcsinfo tiene las expresiones regulares habituales, pero el lado derecho señala a ficheros de plantilla en vez de a programas. Un fichero de plantilla podría ser algo como esto

```
Condición:
Arreglar:
Comentarios:
```

o alguna otra colección de campos que se supone que un desarrollador debe rellenar para formar un informe de cambios válido. La plantilla no es muy útil si todo el mundo hace envíos de cambios usando la opción -m explícitamente, pero muchos desarrolladores prefieren no hacerlo. En lugar de ello, ejecutan

```
floss$ cvs commit
```

y esperan que CVS lance automáticamente un editor de texto (como se especifica en la variable de entorno EDITOR). Ahí escriben un informe de cambios, guardan el fichero y se salen del editor, después de lo cual CVS continúa con el envío.

En ese escenario, se insertaría una plantilla rcsinfo en el editor antes de que el usuario comience a escribir, de forma que se mostrarían los campos junto con un recordatorio para rellenarlos. Entonces, cuando el usuario haga un envío de cambios, se invocará el programa apropiado en '`verifymsg`'. Presumiblemente comprobará que el informe sigue ese formato, y su estado de salida reflejará los resultados de su investigación (con cero indicando éxito).

Como ayuda a los programas de verificación, la ruta a la plantilla del fichero rcsinfo se añade como último argumento en la línea de órdenes de `verifymsg`; de esa forma el programa puede basar su proceso de verificación en la propia plantilla si se desea.

Observe que cuando alguien obtiene una copia de trabajo en una máquina remota, el fichero de plantilla rcsinfo correspondiente se envía al cliente también (se almacena en el subdirectorio CVS/ de la copia de trabajo). Sin embargo esto significa que si se cambia el fichero rcsinfo del servidor después de esto, el cliente no verá los cambios sin volver a obtener una copia del proyecto (con actualizar simplemente no funcionará).

Fíjese también en que en el fichero verifymsg no se admite la palabra clave ALL (aunque DEFAULT sigue valiendo). Esto es para hacer más sencillo saltarse guiones de verificación por defecto y aplicar otros específicos para los subdirectorios.

El fichero taginfo

Lo que loginfo hace con los informes de cambios, taginfo lo hace con las marcas. El lado izquierdo de taginfo está formado por expresiones regulares como siempre, y al lado derecho hay programas. A cada programa se le pasan automáticamente argumentos cuando se invoca una "CVS tag", en este orden:

```
arg 1:          nombre de marca
arg 2:          operación ("añadir" => tag, "mover" => tag -F, "borrar" => \
   tag -d)
arg 3:          repositorio
arg 4, 5, etc:  revisión del fichero [revisión del fichero ...]
```

Si el programa devuelve un resultado distinto de cero, la marca se aborta.

No hemos cubierto la opción -F para marcas antes de ahora, pero es exactamente lo que implica lo de arriba: una forma de mover una marca de una revisión a otra. Por ejemplo, si se añade la marca `Funciona_Bien` a la Revisión 1.7 de un fichero y quiere añadirla en su lugar a la Revisión 1.11, haría esto

```
cvs tag -r 1.11 -F Funciona_Bien foo.c
```

que elimina la marca de 1.7, o dondequiera que estuviera anteriormente en ese fichero, y la pone en 1.11.

El Fichero cvswrappers

El fichero de nombre redundante cvswrappers le da una forma de especificar que ciertos ficheros deberían tratarse como binarios, basado en sus nombres de fichero. CVS no asume que todos los ficheros .jpg sean imágenes JPG, por ejemplo, así que no usa automáticamente -kb cuando añade ficheros JPG. No obstante, algunos proyectos encontrarían muy útil simplemente designar todos los ficheros JPG como binarios. Ésta es la línea de cvswrappers para hacerlo:

```
*.jpg -k 'b'
```

La `b` está separada y entre comillas porque no es el único modo de expansión de palabras clave RCS posible; también podría especificarse `o`, que indica que no se expandan palabras clave con el signo `$`, sino hacer conversión de nueva línea. Sin embargo, `b` es el parámetro más frecuente.

Hay algunos otros modos que se pueden especificar desde el fichero wrappers, pero se utilizan en situaciones tan raras que posiblemente no valga la pena documentarlos aquí (es decir: el autor nunca ha tenido que usarlos). Vea el nodo *Wrappers* en el Cederqvist si siente curiosidad.

El Fichero editinfo

Este fichero está obsoleto, pese a que sigue estando incluido en las distribuciones. Ignórelo.

El Fichero notify

Este fichero se usa junto con las características de `alarmas` de CVS, que se describen en ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩. Nada de ello tendrá sentido hasta que comprenda qué son las alarmas (son una característica útil pero no esencial), así que vea ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩ para obtener detalles sobre este fichero y sobre las alarmas.

El Fichero checkoutlist

Si mira en CVSROOT/, verá qué copias de trabajo de los ficheros existen, junto a sus ficheros de revisión RCS:

```
floss$ ls /usr/local/nuevorepos/CVSROOT
checkoutlist     config,v       history     notify     taginfo
checkoutlist,v   cvswrappers    loginfo     notify,v   taginfo,v
commitinfo       cvswrappers,v  loginfo,v   passwd     verifymsg
commitinfo,v     editinfo       modules     rcsinfo    verifymsg,v
config           editinfo,v     modules,v   rcsinfo,v

floss$
```

CVS presta sólo atención a las versiones de trabajo, no a los ficheros RCS, cuando está buscando una guía sobre cómo comportarse. Por tanto, siempre que haga un envío de cambios de su copia de trabajo de CVSROOT/ (de la que podría obtenerse incluso, después de todo, una copia de trabajo desde otra máquina distinta), CVS actualiza automáticamente todos los ficheros cambiados en el propio repositorio. Sabrá que esto ocurre porque CVS mostrará un mensaje al final de estos envíos:

```
floss$ cvs ci -m "añadidos módulos mp y asub" modules
Checking in modules;
/usr/local/nuevorepos/CVSROOT/modules,v  <--  modules
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
```

CVS se entera automáticamente de lo que pasa con los ficheros estándar de administración, y los reconstruirá en CVSROOT/ cuando sea necesario. Si decide poner ficheros personalizados en CVSROOT/ (como programas o ficheros de plantilla rcsinfo), tendrá que decirle explícitamente a CVS que los trate del mismo modo.

Éste es el propósito del fichero checkoutlist. Tiene un formato distinto al de la mayoría de los ficheros que hemos visto hasta ahora

```
NOMBRE_FICHERO     MENSAJE_DE_ERROR_SI_NO_PUEDE_OBTENERSE_COPIA_DEL_FICHERO
```

por ejemplo,

```
log.pl           imposible obtener copia de / actualizar log.pl en CVSROOT

bugfix.tmpl      imposible obtener copia de / actualizar bugfix.tmpl en CVSROOT
```

Tradicionalmente algunos ficheros de CVSROOT no se someten a control de revisión. Uno de ellos es el fichero *history*, que mantiene un registro en vivo de todas las acciones en el repositorio para usarse con las órdenes `cvshistory` (que lista actividades de marcas, obtenciones de copias y actualizaciones para un fichero o un directorio del proyecto dado). A propósito, si simplemente elimina el fichero '`history`' CVS detendrá servicialmente ese registro.

Nota: a veces el fichero history es la causa de problemas con los permisos, y la forma más fácil de resolverlos es o eliminarlo o hacerlo modificable por todo el mundo.

Otro fichero de administración `no sujeto a revisión` es passwd, dado que obtener una copia suya por la red comprometería las contraseñas (aunque estén cifradas). Tendrá que

decidir basándose en su propia situación de seguridad si quiere añadir passwd a checkoutlist o no; por defecto no está.

Dos notas finales sobre el directorio CVSROOT/: Es posible, si comete un error lo bastante grande, que envíe cambios de un fichero administrativo que esté estropeado de tal forma que impida que se haga cualquier otro envío. Si hace esto, por supuesto que no será capaz de enviar una versión corregida del fichero administrativo!. La solución es ir y editar a mano la copia de trabajo del repositorio del fichero administrativo para corregir el problema; puede que el repositorio entero esté inaccesible hasta que haga esto.

Además, en aras de la seguridad, asegúrese de que en el directorio CVSROOT/ sólo pueden escribir usuarios en quienes confía (con `confianza` quiero decir que confíe tanto en sus intenciones como en su capacidad para no poner en peligro sus contraseñas). Los ficheros '`*info`' le dan a la gente la capacidad de invocar programas en general, así que cualquiera que pueda hacer envíos o editar ficheros en el directorio CVSROOT/ puede en la práctica ejecutar cualquier orden del sistema. Esto es algo que siempre debe tener en mente.

Correos de envío de cambios

Con el fichero loginfo es como se configuran los correos de envío de cambios – correos automáticos que se envían a todos los que trabajan en un proyecto siempre que se realiza un envío de cambios. (Puede que no parezca intuitivo que esto se haga en loginfo en vez de en commitinfo, pero la razón para ello es que se quiere incluir el informe de cambios en el correo). El programa para hacer el envío – '`contrib/log.pl`' en la distribución fuente de CVS – puede instalarse en cualquier parte del sistema. Yo lo pongo en el subdirectorio CVSROOT/ del repositorio, pero es cuestión de gustos.

Puede que necesite editar ligeramente '`log.pl`' para conseguir que funcione en su sistema, posiblemente cambiando la primera línea para que señale a su intérprete Perl, y quizá cambiando la línea

```
$mailcmd = "| Mail -s 'Actualización CVS: $modulepath'";
```

para que llame a su gestor de correo favorito, que puede llamarse `Mail` o no. Una vez que lo haya configurado a su gusto, podrá añadir líneas como estas a su loginfo:

```
listerizer CVSROOT/log.pl %s -f CVSROOT/commitlog -m listerizer@red-bean.com
RoadMail   CVSROOT/log.pl %s -f CVSROOT/commitlog -m roadmail@red-bean.com
bk/*score  CVSROOT/log.pl %s -f CVSROOT/commitlog -m \
                                        bkscore-devel@red-bean.com
```

El `%s` se expande a los nombres de los ficheros afectados por el envío de cambios; la opción -f para '`log.pl`' requiere un nombre de fichero al que el informe de cambios se añadirá (de forma que CVSROOT/commitlog será un fichero siempre creciente de informes de cambios); y el modificador -m admite una dirección de correo electrónico, a la que '`log.pl`' enviará un mensaje sobre el envío de cambios. La dirección es generalmente una lista de correo, pero puede especificar la opción -m tantas veces como sean necesarias en una línea de órdenes para log.pl.

Averiguando más

Aunque este capítulo trata de dar una introducción a la instalación y administración de CVS, me he dejado cosas que o bien se usan muy raramente como para que valga la pena mencionarlas o bien ya están bien documentadas en el manual Cederqvist. La última categoría incluye la puesta a punto de los otros métodos de acceso remoto: RSH/SSH, kserver (Kerberos 4) y GSSAPI (que incluye Kerberos 5 entre otras cosas). Debería notarse que no hay que hacer nada especial para las conexiones con RSH/SSH, aparte de asegurarse de que el usuario en cuestión puede entrar en la máquina del repositorio usando RSH o SSH. Si pueden y CVS está instalado tanto en el cliente como en el servidor, y tienen los permisos adecuados para usar el repositorio directamente desde la máquina servidora, deberían poder acceder al repositorio remotamente por medio del método :ext:.

Las descripciones de algunas características especializadas de CVS se han dejado para capítulos posteriores, para que puedan introducirse en contextos en los que su utilidad es evidente. Pueden encontrarse consejos generales de solución de problemas de CVS en ⟨undefined⟩ [Problemas y Soluciones], page ⟨undefined⟩. Aunque no es necesario leerse el manual Cederqvist entero, debería familiarizarse con él; será una herramienta de referencia de valor incalculable. Si por alguna razón no tiene Info en su máquina y no quiere imprimir el manual, puede hojearlo en línea en `http://durak.org/cvswebsites/doc/` o `http://www.loria.fr/~molli/cvs/doc/cvs_toc.html`.

CVS avanzado

Ahora que hemos cubierto los conceptos básicos sobre el uso de CVS y la administración del repositorio, miraremos cómo CVS puede ser incorporado dentro del proceso de desarrollo. El ciclo de funcionamiento de CVS – obtener (*checkout*), actualizar (*update* entregar (*commit*), actualizar, entregar, y así sucesivamente – fué mostrado en los ejemplos ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩. Este capítulo amplía este ciclo y muestra como CVS puede ser usado para ayudar a los desarrolladores a comunicarse, dar resúmenes del la actividad y la historia del proyecto, fusionar diferentes ramas de desarrollo y ejecutar tareas frecuentes automáticamente. Algunas de las técnicas explicadas introducen nuevas órdenes, pero muchas simplemente indican otra forma mejorada de usar órdenes que ya se han visto.

Alarma (CVS como telefono)

Un beneficio importante de usar CVS en un proyecto es que puede funcionar tanto como un dispositivo de comunicación como para almacenar información sobreel proyecto. Esta sección se centra en cómo se puede usar CVS para que los participantes estén informados de lo que pasa en el projector. Como en otros aspectos de CVS, estas características fomentan la cooperación. Peros los participantes tienen que querer expresamente que se les informe; si la gente elige no usar estas carácteristicas de comunicación no hay nada que CVS puede hacer.

Cómo funcionan las alarmas

En su comportamiento por defecto CVS trata cada copia de trabajo como una caja independiente. Nadie sabe lo que usted está haciendo hasta que entrega sus cambios. Así mismo usted no sabe lo que los demás están haciendo en las suyas; excepto a través de los métodos normales de comunicación, por ejemplo pegando una voz en la oficina: -Oye, voy a trabajar en el fichero.c ahora. Decidme si alguién está trabajando en él para no tener conflictos!

Este método informal funciona en proyectos donde la gente sabe más o menos quién es responsable de qué. Sin embargo, es más dificil cuando hay un gran número de desarrolladores activos en todas las partes del código base y quieren evitarse los conflictos. En estos casos, frecuentemente se pasa al área de responsabilidad de otro compañero y no se puede estar gritando en la oficina ya que el lugar de trabajo es geográficamente disperso.

Una característica de CVS llamada `alarma` proporciona un modo de avisarse entre ellos quién está trabajando en que fichero en un momento dado. Si alguien establece una alarma en un fichero puede ser advertido cuando otro desarrollador empieze a trabajar es ese fichero. La advertencia se envía normalmente por medio de correo electrónico pero es posible usar otros métodos.

Para usar alarmas, hay que modificar uno ó dos ficheros del área de admistración del repositorio, y los desarrolladores tienen que hacer un paso extra en el ciclo ususal de obtención/actualización/entrega. Los cambios en el repositorio son bastante simples: Necesita editar el fichero '`CVSROOT/notify`' para que CVS sepa que advertencias debe realizar. También hay que añadir algunas líneas al fichero '`CVSROOT/users`', que aporta direcciones externas de correo electrónico.

En la copia de trabajo los desarrolladores tienen que decir al CVS qué ficheros debe vigilar para que CVS informe cuando alguién va a editar alguno de esos ficheros. Además uno tiene que decirle a CVS cuando empieza o termina de editar un fichero para que CVS a su vez lo indique a otros desarrolladores que podrían estar vigilando. Las siguientes órdenes son usadas para estos pasos extra.

cvs watch

cvs edit

cvs unedit

El patrón de la orden `watch` se diferencia de otras órdenes comunes en que utiliza subórdenes, como por ejemplo `cvs watch add...`, `cvs watch remove...`, y así.

En el siguiente ejemplo vamos a ver cómo se establecen las alarmas en el repositorio y cómo se usan desde el área del desarrollador. Los dos usuarios de ejemplo, jrandom y qsmith, tienen su propia copia de trabajo que puede estar en diferentes máquinas. Seguimos asumiendo que la variable de entorno $CVSROOT está asignada y por ello no tenemos que pasar la opción -d <REPOS> a ningún comando CVS.

Habilitar alarmas en el repositorio

Primero se debe activar la notificación mediante correo electrónico editando el fichero CVSROOT/notify. Uno de los dos desarrolladores debe hacer esto o el administrador del repositorio si los desarrolladores no tienen permiso para cambiar los ficheros administrativos del repositorio. En cualquier caso la primera cosa es obtener el área administrativa y editar el fichero notify:

```
floss$ cvs -q co CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
U CVSROOT/cvswrappers
U CVSROOT/editinfo
U CVSROOT/loginfo
U CVSROOT/modules
U CVSROOT/notify
U CVSROOT/rcsinfo
U CVSROOT/taginfo
U CVSROOT/verifymsg
floss$ cd CVSROOT
floss$ emacs notify
...
```

Cuando se edita el fichero notify por primera vez nos encontramos con algo como esto:

```
# Versión en castellano
# El fichero 'notify' especifica donde van se envían las notificaciones
# procedentes de alarmas establecidas mediante "cvs watch add" ó "cvs
# edit". La primera entrada de una línea es una expresión regular que se
# compara con el directorio donde el cambio se está haciendo relativo a
# $CVSROOT. Si coincide el resto de la línea es un programa filtro que
# debería contener una ocurrencia %s que indica el usuario a notificar, e
```

```
# informcación de su entrada de datos estándar.
#
# "ALL" o "DEFAULT" puede ser usada en lugar de la expresión regular.
#
# Por ejemplo:
# ALL mail %s -s "notificación de CVS"
```

En realidad todo lo que hay que hacer es descomentar la última línea quitando el carácter `#`. Aunque `notify` proporciana la misma flexibilidad que otros ficheros administrativos a través de las expresiones regulares normalmente no se va usar. La única razón de tener múltiples lineas, cada una con un expresión regular para cada parte del repositorio es si se van usar otros métodos de notificación distintos para cada proyecto. Normalmente la mayoría de los proyectos usan correo electrónico ya que es un buen método de notificación.

Para especificar la notificación mediante correo electrónico, la línea

```
ALL mail %s -s "notificación de CVS"
```

debería funcionar en cualquier Unix estándar. Éste comando hace que las notificaciones o avisos sean enviadas mediante correo electrónico con la línea de `subject notificación de CVS` (La expresión se compara contra cualquier directorio). Cuando se haya descomentado la línea hay que entregar el fichero notify para que el repositoria sea consciente del cambio:

```
floss$ cvs ci -m "establecido notificación por alarma"
cvs commit: Examining .
Checking in notify;
/usr/local/newrepos/CVSROOT/notify,v  <--  notify
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
floss$
```

Editar este fichero es todo lo que hay que hacer para establecer alarmas en el repositorio. Sin embargo si hay desarrolladores trabajando en máquinas remotas es necesario editar el fichero '`CVSROOT/users`' también. La función de este fichero es indicar a CVS a qué direcciones de correo enviar las notificaciones para los usuarios remotos. El formato de cada línea del fichero users sería:

```
CVS_USERNAME:EMAIL_ADDRESS
```

Por ejemplo,

```
qsmith:quentinsmith@farawayplace.com
```

El nombre de usuario al principio de la línea corresponde a un usuario de CVS del fichero '`CVSROOT/password`' (si está presente y el método de acceso por servidor está siendo usado), o el usuario del servidor ejecutando CVS. Siguiendo los dos puntos viene la dirección de correo de ese usuario a la que el CVS enviará las notificaciones.

Desgraciadamente en el momento de la escritura de este documeto el fichero users no existe en la distribución estándar de CVS. Debido a que es un fichero administrativo no sólo se debe crearlo, añadirlo `cvs add ...` y entregarlo `commit ...` de la forma usual sino que hay que añadirlo al fichero '`CVSROOT/checkoutlist`' para que una copia sea mantenida en el repositorio.

Lo siguiente es una sesión de ejemplo:

```
floss$ emacs checkoutlist
  ... (añade la línea para el fichero users) ...
floss$ emacs users
  ... (añade la línea para el usuario qsmith) ...
floss$ cvs add users
floss$ cvs ci -m "añade users a checkoutlist, qsmith a users"
cvs commit: Examining .
Checking in checkoutlist;
/usr/local/newrepos/CVSROOT/checkoutlist,v  <--  checkoutlist
new revision: 1.2; previous revision: 1.1
done
Checking in users;
/usr/local/newrepos/CVSROOT/users,v  <--  users
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
floss$
```

Es posible usar direcciones de correo de formato expandido en '`CVSROOT/usres`', pero hay que tener cuidado en poner todos los espacios en blanco entre comillas. Veamos el siguiente ejemplo

```
qsmith:"Quentin Q. Smith <quentinsmith@farawayplace.com>"
```

o

```
qsmith:'Quentin Q. Smith <quentinsmith@farawayplace.com>'
```

Sin embargo, esto no funcionará:

```
qsmith:"Quentin Q. Smith" <quentinsmith@farawayplace.com>
```

Si hay dudas debería probar ejecutando la orden del fichero notify directamente reemplazando `%s` en

```
mail %s -s "CVS notification"
```

por lo que sigue después de los dos puntos en el fichero users. Si funciona desde el prompt debería hacerlo también en el fichero de usuarios.

Cuando está terminado el fichero checkout debería aparecer como:

```
# El fichero 'checkoutlist' se usa para soportar ficheros adicionales de
# control de versión administrativos de $CVSROOT/CVSROOT, como plantillas.
#
# La primera entrada de una línea es un nombre de fichero que será obtenido
# del correspondiente RCS fichero del directorio $CVSROOT/CVSROOT.
# El resto de la línea será el mensaje de error que aparecerá si el fichero
# no se puede obtener.
#
# Formato del fichero:
#
#       [<espacio en blaco>]<nombre del fichero><espcio en blanco>
<mensaje de error><find de línea>
#
# líneas de comentario enmpiezan con '#'
```

```
users   No ha sido posible obtener users file in CVSROOT.
```

Ya hemos visto como se prepara el repositorio para las alarmas. Pasemos ahora a lo que los desarrolladores tienen que hacer en sus copias de trabajo.

Usando alarmas durante el desarrollo

Primero, un programador obtiene una copia de trabajo y se añade a la lista de vigilantes para alguno de los ficheros del projecto:

```
floss$ whoami
jrandom
floss$ cvs -q co myproj
U myproj/README.txt
U myproj/foo.gif
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
U myproj/b-subdir/random.c
floss$ cd myproj
floss$ cvs watch add hello.c
floss$
```

La última orden, `cvs watch add hello.c`, indica a CVS que nofifique a jramdom si alguien empieza a trabajar en hello.c. O sea que añade jrandom a la lista de alarma de hello.c. Para que CVS pueda notificar tan pronto como sea posible que un fichero se va a editar el usuario tiene que indicárselo a CVS mediante la orden `cvs edit` y el nombre del fichero. CVS no tiene otro modo de saber cuando alguien empieza a trabajar en un fichero. Una ver que se ha obtenido una copia, CVS no es normalmente invocado hasta la siguiente actualización o entrega, la cual sucede después de que el fichero se haya editado:

```
paste$ whoami
qsmith
paste$ cvs -q co myproj
U myproj/README.txt
U myproj/foo.gif
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
U myproj/b-subdir/random.c
paste$ cd myproj
paste$ cvs edit hello.c
paste$ emacs hello.c
...
```

Cuando qsmith ejecuta `cvs edit hello.c`, CVS mira en la lista de alarma de hello.c, ve que jrandom está en ella, y envia un correo a jrandom diciendole que qsmith ha empezado a editar el fichero. El fichero incluso parece haber llegado del mismo qsmith:

```
From: qsmith
Subject: CVS notification
To: jrandom
Date: Sat, 17 Jul 1999 22:14:43 -0500
```

```
myproj hello.c
--
Triggered edit watch on /usr/local/newrepos/myproj
By qsmith

Además cada vez que qsmith (o cualquiera) entrega una nueva revisión de
hello.c, jrandon recivirá otro correo electrónico:


myproj hello.c
--
Triggered commit watch on /usr/local/newrepos/myproj
By qsmith
```

Después de recibir estos correos, jrandom podría querer actualizar su copia de hello.c para ver qué cambios qsmith ha hecho o quizás quisiera enviar un correo a qsmith para saber porqué está trabajando en ese fichero. Nada fuerza a qsmith a recordar hacer un `cvs edit`. Si lo hizo fué porque quizá querría hacerselo saber a jrandom. De cualquier forma, incluso si se olvida hacer un `cvs edit` al hacer una entrega se disparará la notificación. La razón de `cvs edit` es que los advierte a los vigilantes antes de que empizes a trabajar en el fichero y éstos se pueden poner en contacto contigo para resolver el conflicto, antes de que hayas desperdiciado el tiempo.

CVS asume que cualquiera que hace un `cvs edit` quiere ser añadido a la lista de alarma de ese fichero, al menos temporalmente, en caso de que algún otro también lo edite. Cuando qsmith ejecuta `cvs edit`, se convierte en un vigilante de hello.c. Él y jrandom recivirían un notificación si una tercera persona ejecutase `cvs edit` en ese fichero (ó lo entregara).

Sin embargo, CVS también asume que la persona que está editando el fichero quiere estar en la lista mientras está trabajando en él. Esos usuarios serán borrados de la lista cuando hayan hecho su edición del fichero. Si quieren ser vigilantes permanenetes deberán ejecutar `cvs watch add`. CVS asume por defecto que alguién ha terminado la edición cuando él ó ella hace una entrega del fichero.

Cualquiera que está en la lista de vigilancia de un fichero por haber ejecutado `cvs edit` en ese fichero es conocido como *temporary watcher* y es eliminado de la lista cuando entrege un cambio en el fichero. Si quiere editarlo de nuevo tendría que volver a ejecutar `cvs edit`.

La suposición de que la primera entrega acaba la sesión de edición es sólamente una buena predicción, ya que CVS no sabe cuantas entregas la persona necesitará para hacer sus cambios. La predicción es buena para *one-off* cambios – cambios donde hay que arreglar un pequeño error en un fichero y entregarlo. Para ediciones más prolongadas que requieren varias entregas sería bueno que el usuarios se añadiera permanentemente a la lista de vigilantes.

```
paste$ cvs watch add hello.c
paste$ cvs edit hello.c
paste$ emacs hello.c
...
paste$ cvs commit -m "escrito hola en sánscrito"
```

Incluso después de una entraga, qsmith quedará como vigilante de hello.c ya que ejecutó `watch add` sobre él. (A propósito, qsmith no recibirá notificaciones sobre sus propios cambios; sólo los demás. CVS es lo suficientemente listo para no advertirte sobre tus propias acciones.)

Cómo acabar una sesión de edición

Si no quieres hacer una entrega sino acabar tu sesión de edición explicitamente, se puede hacer ejecutando `cvs unedit`:

```
paste$ cvs unedit hello.c
```

Dese cuenta de que además de notificar a los demás vigilantes que ha terminado esta orden ofrece la posibilidad de deshacer todos los cambios que haya hecho en el fichero.

```
paste$ cvs unedit hello.c
hello.c has been modified; revert changes? y
paste$
```

Si usted contesta `y`, CVS deshará todos los cambios y notificará a todos los vigilantes que no seguirá editando el fichero. Si responde `n`, CVS mantiene sus cambios y seguirá registrado como uno de los editores del fichero. Por lo tanto no habrá nofificación; de hecho es como si no hubiera ejecutado `cvs unedit`. Esta posibilidad es un poco tremenda, pero fácil de entender: si declara al mundo que ha terminado su sesión de edición cualquier cambio que no haya entregado antes es porque no tenía intención de guardarlos. Al menos así es como lo ve CVS. No es necesario decir que tenga cuidado.

Controlar qué acciones son vigiladas

Por defecto los vigilantes son noficados ante tres acciones: ediciones, entregas, y terminación de sesión. Sin embargo si usted sólo quiere ser notificado, por ejemplo en las entregas, puede restringir sus notificaciones con el indicador -a de la orden watch (a por action).

```
floss$ cvs watch add -a commit hello.c
```

Es posible pasar el indicador -a dos veces si quiere ser advertido de entregas y notificaciones.

```
floss$ cvs watch add -a edit -a commit hello.c
```

Añadir alarmas con el indicador -a no causará la eliminación de ninguna de sus existentes alarmas. Si estaba vigilando para las tres acciones sobre hello.c, el ejecutar

```
floss$ cvs watch add -a commit hello.c
```

no tiene efecto – usted seguirá teniendo las tres acciones. Para quitar las alarmas hay que hacer lo siguiente

```
floss$ cvs watch remove hello.c
```

Esta orden, por defecto quita las tres clases de acciones. Especificando la acción mediante -a se quitan sólo las alarmas que se especifiquen en la línea de órdenes:

```
floss$ cvs watch remove -a commit hello.c
```

Esto indica que usted desea dejear de recibir notificaciones sobre entregas pero seguirá recibiendo notificaciones sobre ediciones y terminación de edición (asumiendo que tenía vigilancia para estas dos acciones).

Hay dos opciones que se pueden pasar al indicador -a: all ó none (todas ó niguna). *all* significa que las acciones que se vigilarán son las tres antes mencionadas y *none* ninguna de ellas. Ya que el comportamiento por defecto de CVS sin el indicador -a es vigilar todas las acciones y si no se vigila ninguna es como estar fuera de la lista de vigilancia es dificil imaginar una situación en la que que se usen estas dos opciones. Sin embargo, "cvs edit" también usa lleva el indicador -a, y en esta caso puede ser útil especificar *all* ó *none*. Por ejemplo, alguien que va a trabajar brevemente en un fichero quizá no quiera ser notificado sobre lo que están haciendo otros en ese fichero. La orden

```
paste$ whoami
qsmith
paste$ cvs edit -a none README.txt
```

hace que vigilantes de README.txt sean notificados de que qsmith va a trabajar en él, pero qsmith no será un vigilante temporal de README.txt durante su sesión de edición ya que pidió explicitamente no vigilar ninguna acción.

Observe que esto sólo afecta a lo que usted está vigilando con la orden `cvs watch`. Usted puede dejar de vigilar cualquier fichero pero esto no afectará a las alarmas de los demás.

Encontrar quién vigila qué

Alguna vez puede interesarle saber quien está vigilando o editando un fichero sin antes de ejecutar `cvs edit` o ver quién está editando qué sin añadirse a ninguna lista de vigilancia. O podría haber olvidado su propio estatus. Después de haber establecido algunas alarmas y haber entregado algunos ficheros es fácil saber que está uno vigilando y editando.

CVS proporciona dos comandos para mostrar quién está vigilando y editando qué ficheros – `cvs watchers` y `cvs editors`:

```
floss$ whoami
jrandom
floss$ cvs watch add hello.c
floss$ cvs watchers hello.c
hello.c jrandom  edit unedit  commit
floss$ cvs watch remove -a unedit hello.c
floss$ cvs watchers hello.c
hello.c jrandom  edit commit
floss$ cvs watch add README.txt
floss$ cvs watchers
README.txt      jrandom edit    unedit  commit
hello.c jrandom edit    commit
floss$
```

Observe que la última orden `cvs watchers` no especifica ningún fichero y así muestra los vigilantes para todos los ficheros que estén siendo vigilados.

Todas las órdenes `watch` y `edit` tienen en común con otras órdenes CVS esta característica. Si usted especifica nombres de ficheros estas órdenes actuan sobre ellos. Si se especifican nombres de directorio actuan sobre cada fichero de ese directorio y sus subdirectorios. Si no especifica nada, actuan sobre el directorio actual y sus subdirectorios. Siguiendo con la sesión del ejemplo anterior:

```
floss$ cvs watch add a-subdir/whatever.c
```

```
floss$ cvs watchers
README.txt      jrandom edit    unedit  commit
hello.c jrandom edit    commit
a-subdir/whatever.c     jrandom edit    unedit  commit
floss$ cvs watch add
floss$ cvs watchers
README.txt      jrandom edit    unedit  commit
foo.gif jrandom edit    unedit  commit
hello.c jrandom edit    commit  unedit
a-subdir/whatever.c     jrandom edit    unedit  commit
a-subdir/subsubdir/fish.c       jrandom edit    unedit  commit
b-subdir/random.c       jrandom edit    unedit  commit
floss$
```

El penúltimo comando hizó a jrandom un vigilante de todos los ficheros del proyecto y el último mostró cada lista de cada fichero del proyecto. La salida de `cvs watchers` puede que no perfectamente alineado por columnas debido a que se mezclan tabuladores con información de longitud variable, pero el formateo de las líneas es consistente.

```
[FILENAME] [espacio en blanco] WATCHER [espacio en blanco]
ACTIONS-BEING-WATCHED...
```

Observe qué pasa cuando qsmith empieza a editar uno de los ficheros:

```
paste$ cvs edit hello.c
paste$ cvs watchers
README.txt      jrandom edit    unedit  commit
foo.gif jrandom edit    unedit  commit
hello.c jrandom edit    commit  unedit
       qsmith  tedit   tunedit tcommit
a-subdir/whatever.c     jrandom edit    unedit  commit
a-subdir/subsubdir/fish.c       jrandom edit    unedit  commit
b-subdir/random.c       jrandom edit    unedit  commit
```

El fichero hello.c tiene un nuevo vigilante: qsmith (observe que el nombre del fichero no se repite sino que se deja un espacio en blanco al principio de la línea; esto es importante si usted alguna vez escribiera un programa que compila la salida de la orden). Ya que está editando el fichero hello.c, qsmith tendrá una *temporary watch alarma temporal* sobre el fichero, la cual se terminará cuando haga una entrega sobre éste. El prefijo `t` delante de cada acción indica que son alarmas termporales. Si qsmith se añade a la lista de vigilantes regulares de hello.c

```
paste$ cvs watch add hello.c
README.txt      jrandom edit    unedit  commit
foo.gif jrandom edit    unedit  commit
hello.c jrandom edit    commit  unedit
       qsmith  tedit   tunedit tcommit edit    unedit  commit
a-subdir/whatever.c     jrandom edit    unedit  commit
a-subdir/subsubdir/fish.c       jrandom edit    unedit  commit
b-subdir/random.c       jrandom edit    unedit  commit
```

se encontrará a la vez como un vigilante temporal y un vigilante permanente. Se puede decir que el estatus de vigilante permanente sobrepasa al temporal. Entonces la línea sería como:

```
qsmith  edit    unedit  commit
```

Sin embargo, CVS no puede reemplazar las alarmas temporales porque sabe que orden ocurren las acciones. Se quitará qsmith de la lista permanente de vigilancia antes de acabar su sesión de edición?, o acabará sus ediciones siendo todavía un vigilante?. En el primer caso las acciones `edit` / `unedit` / `commit` desaparecen mientras que `tedit` / `tunedit` / `tcommit` permanecen; en el segundo caso ocurre lo contrario. De cualquier forma esto no será de gran importancia. Casi siempre lo que usted hará es ejecutar

```
floss$ cvs watchers
```

o

```
floss$ cvs editors
```

desde el nivel más alto para ver quién está haciendo qué. No necesita conocer los detalles de quien está vigilando que acciones: lo importante son las personas y los ficheros.

Recomendar a la gente usar alarmas

Habrá observado que el funcionamiento de las alarmas depende, en última instancia de la colaboración de todos los desarrolladores. Si alguien empieza a editar un fichero sin ejecutar `cvs edit`, nadie lo sabrá hasta que los cambios se entreguen. Como `cvs edit` se usa a un nivel superior y no está dentro de la rutina normal de desarrollo las personas pueden olvidarse fácilmente de hacerlo.

Aunque CVS no puede forzar a alguien a usar `cvs edit`, tiene un mecanismo que es permite a recordar a los usuarios a hacerlo; la orden `watch on`:

```
floss$ cvs -q co myproj
U myproj/README.txt
U myproj/foo.gif
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
U myproj/b-subdir/random.c
floss$ cd myproj
floss$ cvs watch on hello.c
floss$
```

Ejecutando `cvs watch` sobre hello.c, jrandom hace que futuras obtenciones de copias de myproj hagan que hello.c sea de sólo lectura en la copia de trabajo. Cuando qsmith intenta trabajar sobre él comprobará que es de sólo lectura y se le recordará que debe ejecutar primero `cvs edit`:

```
paste$ cvs -q co myproj
U myproj/README.txt
U myproj/foo.gif
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
U myproj/b-subdir/random.c
paste$ cd myproj
paste$ ls -l
total 6
```

```
drwxr-xr-x   2 qsmith    users        1024 Jul 19 01:06 CVS/
-rw-r--r--   1 qsmith    users          38 Jul 12 11:28 README.txt
drwxr-xr-x   4 qsmith    users        1024 Jul 19 01:06 a-subdir/
drwxr-xr-x   3 qsmith    users        1024 Jul 19 01:06 b-subdir/
-rw-r--r--   1 qsmith    users         673 Jun 20 22:47 foo.gif
-r--r--r--   1 qsmith    users         188 Jul 18 01:20 hello.c
paste$
```

Cuando lo ha hecho, el fichero será de lectura-escritura. Entonces lo podrá editar y cuando haga entregas, pasa a modo de sólo lectura:

```
paste$ cvs edit hello.c
paste$ ls -l hello.c
-rw-r--r--   1 qsmith    users         188 Jul 18 01:20 hello.c
paste$ emacs hello.c
  ...
paste$ cvs commit -m "decir hello en arameo" hello.c
Checking in hello.c;
/usr/local/newrepos/myproj/hello.c,v  <--  hello.c
new revision: 1.12; previous revision: 1.11
done
paste$ ls -l hello.c
-r--r--r--   1 qsmith    users         210 Jul 19 01:12 hello.c
paste$
```

Al hacer esta edición y entrega se enviarán notificaciones a todos los vigilantes de hello.c. Observe que jranadom no es necesariamente uno de ellos. Al ejecutar `cvs watch` sobre hello.c jrandom no se añadió asimismo a la lista de vigilancia para ese fichero; simplemente especificó qe se deberían obtener copias en modo de sólo lectura. Las personas que quieren vigilar un fichero deben añadirse a la lista de vigilancia. CVS no puede hacer nada sobre esto.

Establecer alarmas en un simple fichero sería la excepción. Generalmente es más común establecer las alarmas sobre un proyecto:

```
floss$ cvs -q co myproj
U myproj/README.txt
U myproj/foo.gif
U myproj/hello.c
U myproj/a-subdir/whatever.c
U myproj/a-subdir/subsubdir/fish.c
U myproj/b-subdir/random.c
floss$ cd myproj
floss$ cvs watch on
floss$
```

Esta acción equivale a anunciar una potitica de decisión para todo el proyecto: "*Por favor, use cvs edit para advertir a los vigilantes en que está trabajando y vigile cuantos ficheros le interesen o sean de su responsabilidad.*" Cada fichero del proyecto será actualizado en modo de sólo lectura, y así a la gente se le recordará que se espera que usen `cvs edit` antes de trabajar en algo.

Curiosamente, aunque obtenciones de copias de ficheros vigilados se hacen en modo de sólo lectura, las actualizaciones no lo son. Si qsmith ha obtenido una copia de trabajo

antes de que jrandom hiciera `cvs watch` sus ficheros serían le lectura-escritura quedando así incluso después de hacer actualizaciones. Sin embargo, cualquier fichero que entregue después de que jrandom estableciera las alarmas será de sólo lectura. Si jrandom quita las alarmas:

```
floss$ cvs watch off
```

los ficheros de sólo lectura de qsmith no se convierte mágicamente en ficheros de lectura-escritura. Por otro lado, después de hacer una entrega volverán a modo lectura-escritura de nuevo (como si las alarmas todavía estuviaran puestas).

Observe que qsmith podría, si fuese malicioso, hacer los ficheros de su copia de trabajo escribibles usando la orden estándar de Unix `chmod` saltándose `cvs edit` por completo.

```
paste$ chmod u+w hello.c
```

o si quisiera hacerlo todo en una pasada.

```
paste$ chmod -R u+w .
```

No hay nada que CVS pueda hacer sobre esto. Las copias de trabajo son por su propia naturaleza privadas. Las alarmas pueden permitir su escrutinio al público tanto como lo permita el desarrollador. Sólamente cuando un desarrollador hace algo que afecte al repositorio (como una entrega) su privacidad se pierde incondicinalmente.

La relación entre `watch add`, `watch remove`, `watch on` y `watch off` podría parecer confusa. Para aclarar esto resumamos el esquema general: `add` y `remove` sirven para añadir o quitar usuarios de la lista de vigilancia de un fichero; no tiene nada que ver con que los ficheros sean de sólo lectura al obtener la copia de trabajo o después de la entrega. `on` y `off` sirven para los permisos de los ficheros. No tienen nada que ver con la lista de vigilancia; más bien son herramientas par ayudar a los desarrolladores a recordar la política de alarmas haciendo que los ficheros de la copia de trabajo sean de sólo lectura.

Todo esto parece un poco inconsistente. De algún modo el usar alarmas parece ir en contra de la esencia de CVS. Se desvía del universo ideal de múltiples desarrrolladores editando libremente en sus copias de trabajo, ocultos unos de otros hasta que hacen una entrega. Con las alarmas CVS da a los desarrolladores atajos para informarse mutuamente lo que pasa en sus copias de trabajo. Sin embargo no tiene forma de imponer las políticas de observación ni un concepto de que constituye una sesión de edición. Aun así las alarmas pueden ser útiles en ciertas ocasiones.

Cómo aparecen las alarmas en el repositorio

Para acabar con las cajas negras y los misterios sin solución vamos a hacer una rápida mirada a cómo las alarmas son implementadas en el repositorio. Será rápido ya que no es agradable.

Cuando usted establece una alarma

```
floss$ pwd
/home/jrandom/myproj
floss$ cvs watch add hello.c
floss$ cvs watchers
hello.c jrandom edit    unedit  commit
floss$
```

CVS la guarda en un fichero especial, '`CVS/fileattr`', del subdirectorio apropiado del repositorio.

```
floss$ cd /usr/local/newrepos
floss$ ls
CVSROOT/   myproj/
floss$ cd myproj
floss$ ls
CVS/          a-subdir/     foo.gif,v
README.txt,v  b-subdir/     hello.c,v
floss$ cd CVS
floss$ ls
fileattr
floss$ cat fileattr
Fhello.c        _watchers=jrandom>edit+unedit+commit
floss$
```

El hecho de que fileattr sea almacenado en un subdirectorio llamado CVS del repositorio no significa que el repositorio se haya convertido en una copia de trabajo. Simplemente el nombre `CVS` estaba ya reservado en la copia de trabajo para guardar información con lo que no habrá ningún proyecto que necesite un subdirectorio con ese nombre en el repositorio.

No describiré el formato de '`fileattr`' formalmente; se puede ver bastante bien estudiando los cambios que ocurren en él entre orden y orden.

```
floss$ cvs watch add hello.c
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
Fhello.c        _watchers=jrandom>edit+unedit+commit
floss$ cvs watch add README.txt
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
Fhello.c        _watchers=jrandom>edit+unedit+commit
FREADME.txt     _watchers=jrandom>edit+unedit+commit
floss$ cvs watch on hello.c
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
Fhello.c        _watchers=jrandom>edit+unedit+commit;_watched=
FREADME.txt     _watchers=jrandom>edit+unedit+commit
floss$ cvs watch remove hello.c
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
Fhello.c        _watched=
FREADME.txt     _watchers=jrandom>edit+unedit+commit
floss$ cvs watch off hello.c
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
FREADME.txt     _watchers=jrandom>edit+unedit+commit
floss$
```

Registros de edición son almacenados en fileatrr también. Esto es lo que pasa cuando qsmith se añade asimismo como un editor.

```
paste$ cvs edit hello.c

floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
Fhello.c        _watched=;_editors=qsmith>Tue Jul 20 04:53:23 1999 GMT+floss\
+/home/qsmith/myproj;_watchers=qsmith>tedit+tunedit+tcommit
FREADME.txt     _watchers=jrandom>edit+unedit+commit
```

Finalmente, observe que CVS quita le fichero fileattr y subdirectorio CVS cuando no hay más vigilantes o editores para cualquier fichero en ese directorio:

```
paste$ cvs unedit

floss$ cvs watch off
floss$ cvs watch remove
floss$ cat /usr/local/newrepos/myproj/CVS/fileattr
cat: /usr/local/newrepos/myproj/CVS/fileattr: No such file or directory
floss$
```

Debe quedar claro depués de esta breve exposición que los detalles del tratamiento del formato de fileattr se dejan a CVS. La principal razón para tener una comprensión básica del formato, además de la satisfacción inherente de saber que es lo que ocurre detrás de las cortinas, es si usted va a escribir una extensión a las alarmas de CVS, o está depurando algún error en ellas. Es suficiente con que no se alarme si ve CVS/ subdirectorios apareciendo y desapareciendo de su repositorio. Son los únicos lugares seguros que CVS tiene para almacenar meta-información como las listas de vigilancia.

Mensajes log y correos electrónicos a la entrega

Los correos electrónicos a la entrega son enviados cuando se hace una entrega y muestran el mensaje log y los ficheros involucrados en ésta. Normalmente van para todos los partipantes en el proyecto y algunas veces para otras partes interesadas. Los detalles para establecer correos a la entrega son cubiertos en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩, por lo que no los repetiremos aquí. He observado, sin embargo, algunos efectos inesperados en los proyectos. Efectos que usted debería tener en cuenta si estable correos electrónicos a la entrega.

Prepárese para que la mayoría de los mensajes sean ignorados. Que se lean o no dependerá, al menos en parte, en la frecuencia con que se hagan entregas en su proyecto. Entregan los desarrolladores un cambio grande al final de día, o hacen pequeños cambios durante la jornada? En el último caso los desarrolladores recibirán más mensajes y es problabe que presten menos atención a ellos.

Esto no significa que los mensajes no son útiles, sino que no debe contar con que cada persona va a leer cada mensaje. Es un modo conveniente de para las personas de mantener información de quién está haciendo qué (sin la intrusión de las alarmas). Cuando los correos van a una lista de correos pública subscribible son un mecanismo maravilloso para dar a usuarios interesados (y futuros desarrolladores) una oportunidad para ver lo que ocurre en el código a nivel básico.

Podría tener un desarrollador designado para vigilar todos los mensajes log y tener un visión general de la actividad en todo el proyecto (por supuesto, un buen jefe de proyectos estará probablemente haciendo esto). Si hay una clara división de responsabilidades, es decir, ciertos desarrolladores están a cargo de algún subdirectorio del proyecto, usted podría usar algún lenguaje interpretado sobre CVSROOT/loginfo para que cada responsable reciba notificaciones especiales de cambios hechos en su área. Esto aseguraría que los desarrolladores leen al menos los correos involucrados en sus subdirectorios.

Un interesante efecto que surge cuando los correos no son ignorados. La gente empieza a usarlos como un método de comunicación en tiempo real. Ésta es la clase de mensaje log que podría resultar:

```
Acabada la forma de retroalimentación; arreglados los colores de las
```

```
fuentes y el fondo. Bien, alguien quiere almorzar en 'Los claveles'?
```

No hay nada malo en esto y hace los mensajes log más entretenidos cuando se repasan más tarde. Sin embargo hay que ser consciente de que los mensajes log se guardan para siempre en el historial del proyecto. Por ejemplo, quejarse de las especificaciones de un cliente es un pasatiempo entre los programadores; no es dificil imaginar a alguien entregando un mensaje log como éste, sabiendo que otros programadores lo verán en su correo:

```
Cambiar los cuatro digitos del año por dos en la salida. Lo que el
cliente pide, el cliente recibe, no importa lo estúpido y malo que sea.
```

Esto hace los correos más divertidos pero, qué ocurriría si el cliente revisara los mensajes log? (Apostaría a que debido a esto más de un sitio ha configurado su CVSROOT/loginfo para que invoque scripts que matengan libre de palabras ofensivas los mensajes log.)

El efecto global de los correos a la entraga sería que la gente no escribe mensajes log demasiandos cortos o complicados, lo cual está bien. Sin embargo hay que recordar que su audiencia no son sólo la gente que recibe los correos sino cualquiera que pudiera leer estos mensajes log.

Cambiar un mensaje log después de una entrega

Por si alguien escribe un mensaje log deplorable, CVS permite que éste se puede reescribier después de que se haya entregado. Se hace con el indicador -m de la orden admit (esta orden se verá en más detalle más adelante) y permite cambiar un mensaje log (por revisión, por fichero) cada vez. Así es como funciona:

```
floss$ cvs admin -m 1.7:"Cambiar cuatro dígitos del año por dos en la
salida." date.c
RCS file: /usr/local/newrepos/someproj/date.c,v
done
floss$
```

El mensaje ofensivo original que fue entregado en la revisión 1.7 ha sido reemplazado por uno más inocente (aunque también más soso). No olvide los dos puntos separando el número de revisión del nuevo mensaje log.

Si el mensaje original fue entregado en múltiples fichros, tendrá que ejecurar `cvs admit` para cada uno de ellos, porque el número de revisión será distinto en cada fichero. Así, esta es una de las pocas órdenes en las que CVS requiere que se pase como argumento el nombre de un fichero:

```
floss$ cvs admin -m 1.2:"mensaje log muy aburrido" hello.c REAME.txt foo.gif
cvs admin: while processing more than one file:
cvs [admin aborted]: attempt to specify a numeric revision
floss$
```

Obtendría el mismo error si no le pasa ningún nombre de fichero. Parace confuso pero la razón es que CVS asumiría como argumentos implícitos todos los ficheros de directorio actual.

```
floss$ cvs admin -m 1.2:"mensaje log muy aburrido"
cvs admin: while processing more than one file:
cvs [admin aborted]: attempt to specify a numeric revision
floss$
```

Desgraciadamente esto lo encontramos a menudo en los mensajes de error. Usted debe ver las cosas desde el punto de vista de CVS para que los mensajes tengan sentido.

Invocar `admin -m` cambia el historial del proyecto por lo que debe usarlo con cuidado. No habrá forma de saber que un mensaje log fue cambiado alguna vez. Parecerá como si esa revisión fue entregada con el nuevo mensaje. No quedará huella del mensaje antiguo en ningún sitio (a no ser que salve el correo que se entregó la primera vez).

Aunque por su nombre pueda parecer que sólo administradores designados de CVS pueden usarlo cualquiera puede ejecutar `cvs admin` si tiene acceso de escritura en el proyecto. Sin embargo, es mejor usarlo con cuidado; la habilidad de cambiar el historial de un proyecto es poca comparado con el daño potencial que se puede hacer. Vea ⟨undefined⟩ [Referencia de CVS], page ⟨undefined⟩ para saber más sobre `admin` y también como restringir su uso.

Deshacerse de una copia de trabajo

En un uso típico de CVS, el modo de deshacerse del directorio que contiene una copia de trabajo es quitarlo como se haría con cualquier árbol de directorios:

```
paste$ rm -rf myproj
```

Sin embargo al hacerlo de esta manera otros desarrolladores no sabrán que ha dejado de trabajar en él. CVS proporciona una orden para dejar una copia de trabajor explícitamente. Piense de un lanzamiento como lo contrario de una entrega; usted le dice al repositorio que ya ha hecho su trabajo con la copia de trabajo. Como la entrega, el lanzamiento es invocado desde el directorio padre del arbol.

```
paste$ pwd
/home/qsmith/myproj
paste$ cd ..
paste$ ls
myproj
paste$ cvs release myproj
You have [0] altered files in this repository.
Are you sure you want to release directory 'myproj': y
paste$
```

(Usted tiene [0] ficheros alterados en este repositorio, Está seguro de que quiere lanzarlos (y borrar) el directorio 'myproj': si

Por ahora la versión 1.10.6 no permite que la orden `release` deduzca la localización del repositorio examinando la copia de trabajo ya que `release` es invocado fuera de la copia de trabajo no dentro de ella. Usted debe pasar la opición global `-d <REPOS>` o asegurarse que la variable de entorno CVSROOT está correcta. (Esto se podría arreglar en futuras versiones.)

Cederqvist afirma que si se usa `release` en vez de borrar el directorio de trabajo, la gente que vigila los ficheros liberados será notificada como si hubieran ejecutado `unedit`. Lo he probado experimentalmente y parece que no es verdad.

Historial – Un resumen de la actividad del repositorio

En ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ comenté brebemente la orden `history`. Ésta orden muestra un resumen de todos las obtenciones *checkouts*, entregas *commits*, actualizaciones *updates*, etiquetas *rtags*, y entregas finales *releases* hechos en el repositorio (al menos, desde que `logging` fue activado mediante la creación del fichero CVSROOT/history en el repositorio). Puede controlar el formato y contenidos del resumen con varias optciones.

El primer paso es asegurarse que `logging` esta activo en su repositorio. El administrador del repositorio debe asegurarse de que existe un fichero history

```
floss$ cd /usr/local/newrepos/CVSROOT
floss$ ls -l history
ls: history: No such file or directory
floss$
```

y si no exite crearlo de la siguiente forma:

```
floss$ touch history
floss$ ls -l history
-rw-r--r--   1 jrandom   cvs           0 Jul 22 14:57 history
floss$
```

Este para el historial, `history` debe ser escribible por cualquiera que use el repositorio ya que de otra forma se obtendrá un error cada vez que ejecute alguna orden de CVS que modifique ese fichero. La foram más simple es hacer el fichero escribible por todo el mundo:

```
floss$ chmod a+rw history
floss$ ls -l history
-rw-rw-rw-   1 jrandom   cvs           0 Jul 22 14:57 history
floss$
```

Si el repositorio fue creado con la orden `cvs init`, el fichero `history` ya existirá. Tal vez que tuviese que arreglar los permisos de escritura.

Se asume en el resto de los ejemplos que `history logging` se ha activado durante un tiempo y por lo tanto ha habido tiempo para alguna información se haya acumulado en el historial (fichero `history`).

La salida de `cvs history` es en cierta forma dificil (probablemente se creó pensando en que se tratataría mediante otros programas y no mediante personas, aunque con un poco de estudio se puede leer). Ejecutémoslo y veamos lo que obtenemos.

```
paste$ pwd
/home/qsmith/myproj
paste$ cvs history -e -a
O 07/25 15:14 +0000 qsmith  myproj =mp=     ~/*
M 07/25 15:16 +0000 qsmith  1.14 hello.c    myproj == ~/mp
U 07/25 15:21 +0000 qsmith  1.14 README.txt myproj == ~/mp
G 07/25 15:21 +0000 qsmith  1.15 hello.c    myproj == ~/mp
A 07/25 15:22 +0000 qsmith  1.1  goodbye.c  myproj == ~/mp
M 07/25 15:23 +0000 qsmith  1.16 hello.c    myproj == ~/mp
M 07/25 15:26 +0000 qsmith  1.17 hello.c    myproj == ~/mp
U 07/25 15:29 +0000 qsmith  1.2  goodbye.c  myproj == ~/mp
G 07/25 15:29 +0000 qsmith  1.18 hello.c    myproj == ~/mp
```

```
M 07/25 15:30 +0000 qsmith  1.19 hello.c    myproj == ~/mp
O 07/23 03:45 +0000 jrandom myproj =myproj= ~/src/*
F 07/23 03:48 +0000 jrandom        =myproj= ~/src/*
F 07/23 04:06 +0000 jrandom        =myproj= ~/src/*
M 07/25 15:12 +0000 jrandom 1.13 README.txt myproj == ~/src/myproj
U 07/25 15:17 +0000 jrandom 1.14 hello.c    myproj == ~/src/myproj
M 07/25 15:18 +0000 jrandom 1.14 README.txt myproj == ~/src/myproj
M 07/25 15:18 +0000 jrandom 1.15 hello.c    myproj == ~/src/myproj
U 07/25 15:23 +0000 jrandom 1.1  goodbye.c  myproj == ~/src/myproj
U 07/25 15:23 +0000 jrandom 1.16 hello.c    myproj == ~/src/myproj
U 07/25 15:26 +0000 jrandom 1.1  goodbye.c  myproj == ~/src/myproj
G 07/25 15:26 +0000 jrandom 1.17 hello.c    myproj == ~/src/myproj
M 07/25 15:27 +0000 jrandom 1.18 hello.c    myproj == ~/src/myproj
C 07/25 15:30 +0000 jrandom 1.19 hello.c    myproj == ~/src/myproj
M 07/25 15:31 +0000 jrandom 1.20 hello.c    myproj == ~/src/myproj
M 07/25 16:29 +0000 jrandom 1.3  whatever.c myproj/a-subdir == ~/src/myproj
paste$
```

No está claro?

Antes de examinar la salida, observe que la orden incluye dos opciones: -e y -a. Cuando usted ejecuta `histoy`, casi siempre le pasará opciones para indicar que datos y en que formato los verá. En esto difiere de la mayoria de las órdenes de CVS, que normalmente hacen cosas útiles sin necesidad de opciones. En este ejemplo, los dos indicadores significan respectivamente "todas las cosas" (del inglés *everything*), que muestra todas las claves de eventos que han ocurrido, y "todo" (del inglés *all*), por todos los usuarios.

Otro modo en que `history` se diferencia de otros comandos es que, aunque normalmente se invoca dentro de una copia de trabajo, no sólo está restringida su salida a esa copia del projecto sino que también muestra todo el historial de eventos de todos los proyectos del repositorio. La copia de trabajo sólo indica a CVS desde que repositorio conseguir los datos del historial. (En el ejemplo anterior, los únicos datos de historial de ese repositorio son los del proyecto `myproj`, por lo tanto eso es lo que vemos.)

El formato general de salida es:

```
CÓDIGO FECHA_Y_HORA USUARIO [REVISION] [FICHERO] DIRECTORIO_DEL_REPOSITORIO
NOMBRE_DEL_DIRECTORIO_DE_TRABAJO
```

Aunque la salida de esta orden fue diseñado para ser compacto y usado como entrado por otros programas, CVS le da bastante control sobre su contenido. Las opciones mostradas en la Tabla 6.2 controlan sobre que tipos de eventos se informa.

Tabla 6.1 Significado del código de las letras.

Letra	Significado
O	Obtener
T	Tag
F	Entrega final
W	Actualizar (no de un fichero de usuario, eliminación en las entradas del fichero)
U	Actualizar (fichero sobrescribe un fichero de usuario no modificado)

```
G Actualizar (fichero fusionado exitosamente con un fichero
              modificado de usuario)
C Actualizar (fichero fusionado, pero existen conflictos con
              fichero de usuario)
M Entregar (de un fichero modificado)
A Entregar (un fichero añadido)
R Entregar (el borrado de un fichero)
E Exportar
```

Tabla 6.2 Opciones de filtrado sobre tipo de evento.

Opción	Significado
-m MODULO	Muestra eventos del historial que afecten a MODULO
-c	Muestra las entregas.
-o	Muestra las obtenciones.
-T	Muestra los tag.
-x CODE(S)	Muestra los eventos de tipo CODE (uno o más de OTFWUGCMARE).
-e	Muestra todos los eventos. Una ver seleccionados los tipos de eventos se puede filtrar más con las opciones de la Tabla 6.3.

Tabla 6.3 Opciones a filtrar por el usuario.

Opción	Significado
-a	Muestra las acciones hechas por todos los usuarios
-w	Muestra sólo las acciones hechas dentro de la copia de trabajo.
-l	Muestra sólo la última vez que este usuario realizó la acción
-u USUARIO	Muestra los registros para USUARIO

Anotaciones – Sumario detallado de la de la actividad del proyecto

La orden annotaded

Mientras que la orden `history` da una visión general sobre la actividad del proyecto, *annotate* es un modo de acercarse con más detalle a esa visión. Con `annotate` usted puede ver quién fue la última perosna que tocó cada línea de un fichro, y en que revisión se hizó.

```
floss$ cvs annotate
Annotations for README.txt
***************
1.14         (jrandom  25-Jul-99): blah
1.13         (jrandom  25-Jul-99): test 3 for history
1.12         (qsmith   19-Jul-99): test 2
1.11         (qsmith   19-Jul-99): test
1.10         (jrandom  12-Jul-99): blah
```

```
1.1          (jrandom  20-Jun-99): Just a test project.
1.4          (jrandom  21-Jun-99): yeah.
1.5          (jrandom  21-Jun-99): nope.
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.15         (jrandom  25-Jul-99):   /* another test for history */
1.13         (qsmith   19-Jul-99):   /* random change number two */
1.10         (jrandom  12-Jul-99):   /* test */
1.21         (jrandom  25-Jul-99):   printf ("Hellooo, world!\n");
1.3          (jrandom  21-Jun-99):   printf ("hmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.11         (qsmith   18-Jul-99):   /* added this comment */
1.16         (qsmith   25-Jul-99):   /* will merge these changes */
1.18         (jrandom  25-Jul-99):   /* will merge these changes too */
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
Annotations for a-subdir/whatever.c
***************
1.3          (jrandom  25-Jul-99): /* A completely non-empty C file. */
Annotations for a-subdir/subsubdir/fish.c
***************
1.2          (jrandom  25-Jul-99): /* An almost completely empty C file. */
Annotations for b-subdir/random.c
***************
1.1          (jrandom  20-Jun-99): /* A completely empty C file. */
floss$
```

La salida de `annotate` es bastante intuitiva. A la izquiera está el número de revisión, desarrollador, y la fecha en que esa línea fué añadida o modificada. A la derecha está la línea en questión en su actual versión. Como cada línea es comentado se puede ver el contenido entero del fichero a la derecha de la información anotada.

Si especifica un número de revisión, las anotaciones son dadas para esa revisión lo cual quiere decir que se muestran la más reciente modificacione para cada línea a esa o una anterior revisión. Este es probablemente el modo más comunmente usado. Se examina una revision particular de un fichero para determinar que desarrolladores estaban activos en cada parte de un fichero.

En el ejemplo anterior se puede ver que la más reciente revisión de hello.c es la 1.21, en la que jrandom hizo algo en la línea:

```
printf ("Hellooo, world!\n");
```

Un modo de ver lo que ella hizo es usar `diff` de esa revisión con la anterior:

```
floss$ cvs diff -r 1.20 -r 1.21 hello.c
Index: hello.c
===================================================================
```

```
RCS file: /usr/local/newrepos/myproj/hello.c,v
retrieving revision 1.20
retrieving revision 1.21
diff -r1.20 -r1.21
9c9
<   printf ("Hello, world!\n");
--
>   printf ("Hellooo, world!\n");
floss$
```

Otro modo de verlo manteniendo la amplia visión de la actividad de cada uno es comparando las actuales anotaciones con las anotaciones de una revisión anterior:

```
floss$ cvs annotate -r 1.20 hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.15         (jrandom  25-Jul-99):   /* another test for history */
1.13         (qsmith   19-Jul-99):   /* random change number two */
1.10         (jrandom  12-Jul-99):   /* test */
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.3          (jrandom  21-Jun-99):   printf ("hmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.11         (qsmith   18-Jul-99):   /* added this comment */
1.16         (qsmith   25-Jul-99):   /* will merge these changes */
1.18         (jrandom  25-Jul-99):   /* will merge these changes too */
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$
```

Aunque el `diff` muestra los cambios sobre el texto de forma más concisa, la anotación puede ser preferible ya que coloca estos cambios en su contexto histórico al mostrar desde cuando ha estado presente esa línea (en este caso desde la revisión 1.1). Ese conocimiento puede ayudarle a decidir si mirar en los mensajes log para averiguar los motivos del cambio:

```
floss$ cvs log -r 1.21 hello.c
RCS file: /usr/local/newrepos/myproj/hello.c,v
Working file: hello.c
head: 1.21
branch:
locks: strict
access list:
symbolic names:
       random-tag: 1.20
       start: 1.1.1.1
       jrandom: 1.1.1
keyword substitution: kv
total revisions: 22;    selected revisions: 1
```

```
description:
----------------------------
revision 1.21
date: 1999/07/25 20:17:42;  author: jrandom;  state: Exp;  lines: +1 -1
say hello with renewed enthusiasm
=============================================================================
floss$
```

Además de la opción -r, se puede filtrar las anotaciones con la opción -D DATE:

```
floss$ cvs annotate -D "5 weeks ago" hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$ cvs annotate -D "3 weeks ago" hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.3          (jrandom  21-Jun-99):   printf ("hmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$
```

Anotaciones y ramificaciones

Por defecto, las anotaciones muestran siempren la actividad de la rama principal de desarrollo. Incluso aunque se llame desde una copia de trabajo derivada se muestra las anotaciones de la rama principal a menos que se especifique lo contrario. (Dependiendo de su punto de vista esta tendencia de favorecer el tronco principal se puedría considerar un error o una característica.) Puede anotar una ramificación o derivación pasando el nombre de ésta como argumento de -r. He aquí un ejemplo de una copia de trabajo cuyo fichero hello.c está en una derivación llamada `Brancho_Gratuito`, con al menos un cambio entregado en esa rama:

```
floss$ cvs status hello.c
===================================================================
File: hello.c           Status: Up-to-date

  Working revision:    1.10.2.2        Sun Jul 25 21:29:05 1999
```

```
  Repository revision: 1.10.2.2       /usr/local/newrepos/myproj/hello.c,v
  Sticky Tag:          Brancho_Gratuito (branch: 1.10.2)
  Sticky Date:         (none)
  Sticky Options:      (none)

floss$ cvs annotate hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.10         (jrandom  12-Jul-99):   /* test */
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.3          (jrandom  21-Jun-99):   printf ("hmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$ cvs annotate -r Brancho_Gratuito hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.10         (jrandom  12-Jul-99):   /* test */
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.10.2.2     (jrandom  25-Jul-99):   printf ("hmmmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.10.2.1     (jrandom  25-Jul-99):   printf ("added this line");
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$
```

También puede pasar el número de la ramificación:

```
floss$ cvs annotate -r 1.10.2 hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.10         (jrandom  12-Jul-99):   /* test */
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.10.2.2     (jrandom  25-Jul-99):   printf ("hmmmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.10.2.1     (jrandom  25-Jul-99):   printf ("added this line");
```

```
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$
```

o el número de revisión completo de la ramificación:

```
floss$ cvs annotate -r 1.10.2.1 hello.c
Annotations for hello.c
***************
1.1          (jrandom  20-Jun-99): #include <stdio.h>
1.1          (jrandom  20-Jun-99):
1.1          (jrandom  20-Jun-99): void
1.1          (jrandom  20-Jun-99): main ()
1.1          (jrandom  20-Jun-99): {
1.10         (jrandom  12-Jul-99):   /* test */
1.1          (jrandom  20-Jun-99):   printf ("Hello, world!\n");
1.3          (jrandom  21-Jun-99):   printf ("hmmm\n");
1.4          (jrandom  21-Jun-99):   printf ("double hmmm\n");
1.10.2.1     (jrandom  25-Jul-99):   printf ("added this line");
1.2          (jrandom  21-Jun-99):   printf ("Goodbye, world!\n");
1.1          (jrandom  20-Jun-99): }
floss$
```

Si hace esto, recuerde que los números son sólo válidos parar ese fichero particualr. En general es mejor usar el nombre de esa ramificación si ello es posible.

Usando expansión de palabras

Podría recordar una breve mención de `keyword expansion` en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩. Estas palabras especiales de RCS están rodeadas por el signo del dolar, que CVS busca en el texto del fichero y las sustituye por información de revisión. Por ejemplo si un fichero contiene

```
$Author: jfs $
```

entonces cuando se actualize ese fichero a cualquier revisión, CVS lo sustituirá por el nombre de usuario de la persona que realizó la entrega de esa revisión:

```
$Author: jfs $
```

CVS es también consciente de las palabras que han sido sustituidas por lo que estas se pueden actualizar cuando sea apropiado.

Aunque estas palabras no ofrecen información que no puediera ser obtenida mediante otros medios, dan a las personas una forma cómoda de ver los hechos de revisión incluidos en el texto mismo, en vez de tener que invocar alguna orden rara de CVS.

He aquí otros ejemplos de sustitución de palabras:

```
$Date: 2002/12/05 19:10:27 $       ==>  date of last commit, expands to ==>
$Date: 2002/12/05 19:10:27 $

$Id: chapter-6.texi,v 1.4 2002/12/05 19:10:27 jfs Exp $         ==>  filename, revisio
$Id: chapter-6.texi,v 1.4 2002/12/05 19:10:27 jfs Exp $

$Revision: 1.4 $   ==>  exactly what you think it is, expands to ==>
```

```
$Revision: 1.4 $

$Source: /home/cvs/lucas/doc-cvsbook-es/chapter-6.texi,v $     ==> path to correspondi
$Source: /home/cvs/lucas/doc-cvsbook-es/chapter-6.texi,v $

$Log: chapter-6.texi,v $
Revision 1.4  2002/12/05 19:10:27  jfs
Ahora el documento compila incluyendo los acentos en castellano, he seguido
los consejos de
http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
Sin embargo sería conveniente revisar por qué los nombres de nodo que se
referencian no pueden estar con ISO-latin1... problema de TexInfo?

Revision 1.3  2002/12/03 13:04:59  carlosgarcia
Traduccion de main, introduction e index

Revision 1.2  2002/11/27 16:26:56  carlosgarcia
Realizados arreglos para compilación

Revision 1.1  2000/09/16 12:07:53  jjamor
Traduccion del capitulo 6 insertada en la B.de trabajo
        ==>  accumulating log messages for the file, expands to ==>
$Log: chapter-6.texi,v $
Revision 1.4  2002/12/05 19:10:27  jfs
Ahora el documento compila incluyendo los acentos en castellano, he seguido
los consejos de
http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
Sin embargo sería conveniente revisar por qué los nombres de nodo que se
referencian no pueden estar con ISO-latin1... problema de TexInfo?

Revision 1.3  2002/12/03 13:04:59  carlosgarcia
Traduccion de main, introduction e index

Revision 1.2  2002/11/27 16:26:56  carlosgarcia
Realizados arreglos para compilación

Revision 1.1  2000/09/16 12:07:53  jjamor
Traduccion del capitulo 6 insertada en la B.de trabajo

Revision 1.2  1999/07/26 06:47:52  jrandom
...and this is the second log message.

Revision 1.1  1999/07/26 06:39:46  jrandom
This is the first log message...
```

La palabra $Log: chapter-6.texi,v $ La palabra Revision 1.4 2002/12/05 19:10:27 jfs La palabra Ahora el documento compila incluyendo los acentos en castellano, he seguido La palabra los consejos de La palabra http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html La palabra Sin embargo sería conveniente revisar por qué los nombres de nodo que se La palabra referencian no pueden estar con ISO-latin1... problema de TexInfo? La palabra

La palabra Revision 1.3 2002/12/03 13:04:59 carlosgarcia La palabra Traduccion de main, introduction e index La palabra La palabra Revision 1.2 2002/11/27 16:26:56 carlosgarcia La palabra Realizados arreglos para compilación La palabra La palabra Revision 1.1 2000/09/16 12:07:53 jjamor La palabra Traduccion del capitulo 6 insertada en la B.de trabajo La palabra es la única que se exande varias lineas. A diferencia de las otras no reemplaza la antigua sustitución con una nueva, sino que inserta la última sustitución más una línea en blanco justo después de la palabra especial (las previas sustituciones quedan más abajo). Además cualquier texto entre el principio de línea y el $Log es usada como un prefijo para las sustituciones (esto se hace para asegurar que los mensajes log quedan comentados en el código del programa). Por ejemplo, si usted pone esto dentro de un fichero

```
// $Log: chapter-6.texi,v $
// Revision 1.4  2002/12/05 19:10:27  jfs
// Ahora el documento compila incluyendo los acentos en castellano, he seguido
// los consejos de
// http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
// Sin embargo sería conveniente revisar por qué los nombres de nodo que se
// referencian no pueden estar con ISO-latin1... problema de TexInfo?
//
// Revision 1.3  2002/12/03 13:04:59  carlosgarcia
// Traduccion de main, introduction e index
//
// Revision 1.2  2002/11/27 16:26:56  carlosgarcia
// Realizados arreglos para compilación
//
// Revision 1.1  2000/09/16 12:07:53  jjamor
// Traduccion del capitulo 6 insertada en la B.de trabajo
//
```

se sustituirá por esto en la primera entrega:

```
// $Log: chapter-6.texi,v $
// Revision 1.4  2002/12/05 19:10:27  jfs
// Ahora el documento compila incluyendo los acentos en castellano, he seguido
// los consejos de
// http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
// Sin embargo sería conveniente revisar por qué los nombres de nodo que se
// referencian no pueden estar con ISO-latin1... problema de TexInfo?
//
// Revision 1.3  2002/12/03 13:04:59  carlosgarcia
// Traduccion de main, introduction e index
//
// Revision 1.2  2002/11/27 16:26:56  carlosgarcia
// Realizados arreglos para compilación
//
// Revision 1.1  2000/09/16 12:07:53  jjamor
// Traduccion del capitulo 6 insertada en la B.de trabajo
//
// Revision 1.14  1999/07/26 07:03:20  jrandom
// this is the first log message...
```

```
//
```

a esto en la segunda:

```
// $Log: chapter-6.texi,v $
// Revision 1.4  2002/12/05 19:10:27  jfs
// Ahora el documento compila incluyendo los acentos en castellano, he seguido
// los consejos de
// http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
// Sin embargo sería conveniente revisar por qué los nombres de nodo que se
// referencian no pueden estar con ISO-latin1... problema de TexInfo?
//
// Revision 1.3  2002/12/03 13:04:59  carlosgarcia
// Traduccion de main, introduction e index
//
// Revision 1.2  2002/11/27 16:26:56  carlosgarcia
// Realizados arreglos para compilación
//
// Revision 1.1  2000/09/16 12:07:53  jjamor
// Traduccion del capitulo 6 insertada en la B.de trabajo
//
// Revision 1.15  1999/07/26 07:04:40  jrandom
// ...and this is the second log message...
//
// Revision 1.14  1999/07/26 07:03:20  jrandom
// this is the first log message...
//
```

y así sucesivamente:

```
// $Log: chapter-6.texi,v $
// Revision 1.4  2002/12/05 19:10:27  jfs
// Ahora el documento compila incluyendo los acentos en castellano, he seguido
// los consejos de
// http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
// Sin embargo sería conveniente revisar por qué los nombres de nodo que se
// referencian no pueden estar con ISO-latin1... problema de TexInfo?
//
// Revision 1.3  2002/12/03 13:04:59  carlosgarcia
// Traduccion de main, introduction e index
//
// Revision 1.2  2002/11/27 16:26:56  carlosgarcia
// Realizados arreglos para compilación
//
// Revision 1.1  2000/09/16 12:07:53  jjamor
// Traduccion del capitulo 6 insertada en la B.de trabajo
//
// Revision 1.16  1999/07/26 07:05:34  jrandom
// ...and this is the third!
//
// Revision 1.15  1999/07/26 07:04:40  jrandom
// ...and this is the second log message...
```

```
//
// Revision 1.14  1999/07/26 07:03:20  jrandom
// this is the first log message...
//
```

Podría no querer mantener el historial completo en el fichero todo el tiempo; si es así siempre puede quitar las antiguas secciones cuando sea muy grande. Es más conveniente que tener que ejecutar `cvs log`, y podría ser útil en proyectos donde la gente debe leer constantemente los mensajes logs.

Una técnica más común puede ser incluir $Revision: 1.4 $ en un fichero y usarlo como número de versión para el programa. Esto puede funcionar si el proyecto consiste de un fichero o se llevan a cabo muchas entregas finales (releases) y al menos se garantize que uno de los ficheros se ha modificado entre cada entrega. Incluso se pueden usar estas palabras de expansión como un valor en el código de un programa:

```
VERSION = "$Revision: 1.4 $";
```

CVS sustituye esa palabra como cualquier otra; no tiene en cuenta la semántica del lenguaje de programación ni asume que las dobles comillas protegen la cadena de algún modo.

Una lista completa de palabras de sustitución (hay algunas más poco usuales) se encuentra en ⟨undefined⟩ [Referencia de CVS], page ⟨undefined⟩

Salir del limbo (Cómo trabajar con derivaciones y sobrevivir)

Las derivaciones o ramificaciones son una de las más importantes y más fácilmente mal usadas características del CVS. Aislar los cambios arriesgados o perturbadores en una línea de desarrollo separada hasta que se haya estabilizado puede ser inmensamente beneficioso. Sin embargo, si no se usa apropiadamente puede llevar un proyecto a la confusión y al caos, cuando la gente no pierde la cuento sobre qué cambios se han fusionado y cuando se realizaron.

Algunos principios para trabajar con derivaciones

Para trabajar exitosamente con derivaciones, su grupo de desarrollo debería seguir estos principios:

Minimizar el número de derivaciones activas cada vez. Cuantas más derivaciones se esten desarrrollando al mismo tiempo habrá más posibilidad de que existan conflictos cuando se fusionen con la rama principal. En términos prácticos, la forma de conseguir esto es fusionar tan frecuentemente como pueda (cada vez que una derivación está en un punto estable) y volver al desarrollo de la rama principal cuando esto sea viable. Minimizando la cantidad de desarrollo en paralelo es posible estar al tanto de lo que pasa en cada rama y la posibilidad de conflictos se reduce.

Esto no significa minimizar el número absoluto de derivaciones de un proyecto sino el número de ellos en las que se trabajo en un momentod dado.

Minimizar la complejidad – es decir la profundidad – del esquema de sus derivaciones. Hay circunstancias en que es apropiado tener derivaciones de derivaciones pero son muy

raras (usted podría no encontrar una situación como ésta durante toda su vida como programador). El que CVS permita técnicamente que se puedan tener distintos niveles de derivaciones anidadas, y que se pueda fusionar unas con otras, no quiere decir que usted quiera hacerlo. En la mayoría de las situaciones lo mejor es tener las derivaciones sobre la rama principal y fusionar de la derivación al tronco y vuelta a empezar.

Use consistentemente etiquetas para marcar todas los eventos de fusión y ramificación. Idealmente el significado de cada etiqueta y su relación con otras ramificaciones y etiquetas debería quedar claro por su nombre. (Esto quedará más claro cuando veamos los ejemllos.)

Con estos principios en la cabeza veamos un típico escenario de desarrollo con una ramificación. Tendremos jrandom en la rama principal y qsmith en la derivación. Pero tenga en cuento que podría haber múltiples desarrolladores en ambos sitios. El desarrollo normal en cada línea puede involucrar cualquier número de personas; sin embargo el etiquetado y fusión es mejor hacerlos por una persano en cada lado como verá.

Fusionar repetidamente con la rama principal

Supongamos que qsmith necesita hacer desarrollo en una derivación para no desestabilizar la rama principal que comparte con jrandom. El primer paso es crear una rama nueva. Observe como primero qsmith crea una etiqueta normal (no-rama) en ese punto de la rama principal y después crea la derivación:

```
paste$ pwd
/home/qsmith/myproj
paste$ cvs tag Root-of-Exotic_Greetings
cvs tag: Tagging .
T README.txt
T foo.gif
T hello.c
cvs tag: Tagging a-subdir
T a-subdir/whatever.c
cvs tag: Tagging a-subdir/subsubdir
T a-subdir/subsubdir/fish.c
cvs tag: Tagging b-subdir
T b-subdir/random.c
paste$ cvs tag -b Exotic_Greetings-branch
cvs tag: Tagging .
T README.txt
T foo.gif
T hello.c
cvs tag: Tagging a-subdir
T a-subdir/whatever.c
cvs tag: Tagging a-subdir/subsubdir
T a-subdir/subsubdir/fish.c
cvs tag: Tagging b-subdir
T b-subdir/random.c
paste$
```

Etiquetar primero la rama principal podría servir para obtener algún día la rama principal en el momento de que la derivación fue creada. Si tuviese que hacer eso debería haber

un modo de referirse a esa instantánea de la rama principal sin referirse a la derivación. No puede usar la etiqueta de la derivación ya que lo que obtendría es esa derivación no las revisiones que forman la raiz del tronco. El único modo de hacer esto sería hacer una etiqueta de las revisiones de las que sale la derivación. (Alguna gente que esta regla tan fielmente que consideré listarla como "principio número 4 de ramificación: Crear siempre una etiqueta no-derivación en la posición de la derivación." Sin embargo en algunos sitios no se usa y parece que lo hacen bien por lo que es una cuestión de gusto.) De ahora en adelante me referiré a esta etiqueta no-derivación como *etiqueta del punto de derivación*.

Observe que me he adherido a una convención de nombres: La etiqueta del punto de derivación empieza con `Root-of-` (Raiz-de-), y después el nombre, que usará subrayado en vez de guión para separar las palabras. Cuando la derivación es creada su etiqueta acabará con el sufijo `-branch` (rama) que le indicará con sólo mirar el nombre que es una derivación. (La etiqueta del punto de derivación `Root-of-Exotic_Greetings` no incluye el sufijo `-branch` porque no es una derivación.) No tiene que usar esta convención en particular pero desde luego es aconsejable usar alguna.

Por supuesto, he sido extra pedante. En pequeños proyectos donde cada uno sabe quién está haciendo qué y se pueden arreglar fácilmente las confusiones estas convenciones no tienen que ser usadas. El que use la etiqueta del punto de derivación o una estricta convención de nombres para sus etiquetas dependerá de la complejidad del proyecto y su esquema de derivaciones. (No olvide que siempre puede volver atrás más tarde para actualizar viejas etiquetas y usar una nueva convención; obtenga la versión de la vieja etiqueta, añada la nueva etiqueta y borre después la antigua.)

Ahora qsmith puede empezar a trabajar con la derivación:

```
paste$ cvs update -r Exotic_Greetings-branch
cvs update: Updating .
cvs update: Updating a-subdir
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
paste$
```

Hace algunos cambios a un par de ficheros y los entrega en la derivación:

```
paste$ emacs README.txt a-subdir/whatever.c b-subdir/random.c
...
paste$ cvs ci -m "print greeting backwards, etc"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in README.txt;
/usr/local/newrepos/myproj/README.txt,v  <--  README.txt
new revision: 1.14.2.1; previous revision: 1.14
done
Checking in a-subdir/whatever.c;
/usr/local/newrepos/myproj/a-subdir/whatever.c,v  <--  whatever.c
new revision: 1.3.2.1; previous revision: 1.3
done
Checking in b-subdir/random.c;
/usr/local/newrepos/myproj/b-subdir/random.c,v  <--  random.c
```

```
new revision: 1.1.1.1.2.1; previous revision: 1.1.1.1
done
paste$
```

Mientras tanto jrandom sigue trabajando en el tronco. Ella modifica dos o tres ficheros que qsmith tocó. Para ponerlo más dificil haremos sus cambios creen conflictos con el trabajo de qsmith:

```
floss$ emacs README.txt whatever.c
 ...
floss$ cvs ci -m "some very stable changes indeed"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in README.txt;
/usr/local/newrepos/myproj/README.txt,v  <--  README.txt
new revision: 1.15; previous revision: 1.14
done
Checking in a-subdir/whatever.c;
/usr/local/newrepos/myproj/a-subdir/whatever.c,v  <--  whatever.c
new revision: 1.4; previous revision: 1.3
done
floss$
```

El conflicto no es aparente todavía ya que ninguno de los desarrolladores ha intentado hacer la fusión de la derivación con el tronco. Ahora jrandom hace la fusión:

```
floss$ cvs update -j Exotic_Greetings-branch
cvs update: Updating .
RCS file: /usr/local/newrepos/myproj/README.txt,v
retrieving revision 1.14
retrieving revision 1.14.2.1
Merging differences between 1.14 and 1.14.2.1 into README.txt
rcsmerge: warning: conflicts during merge
cvs update: Updating a-subdir
RCS file: /usr/local/newrepos/myproj/a-subdir/whatever.c,v
retrieving revision 1.3
retrieving revision 1.3.2.1
Merging differences between 1.3 and 1.3.2.1 into whatever.c
rcsmerge: warning: conflicts during merge
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
RCS file: /usr/local/newrepos/myproj/b-subdir/random.c,v
retrieving revision 1.1.1.1
retrieving revision 1.1.1.1.2.1
Merging differences between 1.1.1.1 and 1.1.1.1.2.1 into random.c
floss$ cvs update
cvs update: Updating .
C README.txt
cvs update: Updating a-subdir
C a-subdir/whatever.c
```

```
cvs update: Updating a-subdir/subsubdir
cvs update: Updating b-subdir
M b-subdir/random.c
floss$
```

Dos de los ficheros tienen conflictos. No importa, con su saber hacer jarandom resuelve los conflictos, entrega y etiqueta el tronco indicando una fusión con éxito.

```
floss$ emacs README.txt a-subdir/whatever.c
 ...
floss$ cvs ci -m "merged from Exotic_Greetings-branch (conflicts resolved)"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in README.txt;
/usr/local/newrepos/myproj/README.txt,v  <--  README.txt
new revision: 1.16; previous revision: 1.15
done
Checking in a-subdir/whatever.c;
/usr/local/newrepos/myproj/a-subdir/whatever.c,v  <--  whatever.c
new revision: 1.5; previous revision: 1.4
done
Checking in b-subdir/random.c;
/usr/local/newrepos/myproj/b-subdir/random.c,v  <--  random.c
new revision: 1.2; previous revision: 1.1
done
floss$ cvs tag merged-Exotic_Greetings
cvs tag: Tagging .
T README.txt
T foo.gif
T hello.c
cvs tag: Tagging a-subdir
T a-subdir/whatever.c
cvs tag: Tagging a-subdir/subsubdir
T a-subdir/subsubdir/fish.c
cvs tag: Tagging b-subdir
T b-subdir/random.c
floss$
```

Mientras, qsmith no necesita esperar que termine la fusión para continuar el desarrollo si hace una etiqueta del conjunto de cambios que jrandom fusionó (más tarde, jrandom necesitará saber el nombre de esta etiqueta; en general las derivaciones dependen de una frecuente y completa comunicación entre los desarrolladores):

```
paste$ cvs tag Exotic_Greetings-1
cvs tag: Tagging .
T README.txt
T foo.gif
T hello.c
cvs tag: Tagging a-subdir
T a-subdir/whatever.c
```

```
cvs tag: Tagging a-subdir/subsubdir
T a-subdir/subsubdir/fish.c
cvs tag: Tagging b-subdir
T b-subdir/random.c
paste$ emacs a-subdir/whatever.c
 ...
paste$ cvs ci -m "print a randomly capitalized greeting"
cvs commit: Examining .
cvs commit: Examining a-subdir
cvs commit: Examining a-subdir/subsubdir
cvs commit: Examining b-subdir
Checking in a-subdir/whatever.c;
/usr/local/newrepos/myproj/a-subdir/whatever.c,v  <--  whatever.c
new revision: 1.3.2.2; previous revision: 1.3.2.1
done
paste$
```

Y por supuesto cuando qsmith haya hecho sus cambios tendrá que etiquetar:

```
paste$ cvs -q tag Exotic_Greetings-2
T README.txt
T foo.gif
T hello.c
T a-subdir/whatever.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
paste$
```

Mientras todo esto sucede jrandom hace un cambio en un fichero distinto, uno que qsmith no ha tocado en sus ediciones:

```
floss$ emacs README.txt
 ...
floss$ cvs ci -m "Mention new Exotic Greeting features" README.txt
Checking in README.txt;
/usr/local/newrepos/myproj/README.txt,v  <--  README.txt
new revision: 1.17; previous revision: 1.16
done
floss$
```

En este momento qsmith ha entregado un nuevo cambio en su derivación y jrandom ha entregado otro cambio no conflictivo en un fichero distinto del tronco. Observe que sucede cuando jrandom trata de fusionar desde la derivación de nuevo:

```
floss$ cvs -q update -j Exotic_Greetings-branch
RCS file: /usr/local/newrepos/myproj/README.txt,v
retrieving revision 1.14
retrieving revision 1.14.2.1
Merging differences between 1.14 and 1.14.2.1 into README.txt
rcsmerge: warning: conflicts during merge
RCS file: /usr/local/newrepos/myproj/a-subdir/whatever.c,v
retrieving revision 1.3
retrieving revision 1.3.2.2
Merging differences between 1.3 and 1.3.2.2 into whatever.c
```

```
rcsmerge: warning: conflicts during merge
RCS file: /usr/local/newrepos/myproj/b-subdir/random.c,v
retrieving revision 1.1
retrieving revision 1.1.1.1.2.1
Merging differences between 1.1 and 1.1.1.1.2.1 into random.c
floss$ cvs -q update
C README.txt
C a-subdir/whatever.c
floss$
```

Hay conflictos! Esperaba esto?

El problema radica en el significado de fusionar. En ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩ expliqué que cuando usted ejecuta

```
floss$ cvs update -j BRANCH
```

en una copia de trabajo, CVS fusiona en la copia de trabajo las diferencias entre la raiz BRANCH y su estado actual. El problema con este comportamiento es que, en esta situación, la mayoría de esos cambios ya habían sido incorporados al tronco la primera vez que jrandom hizo una fusión. Cuando CVS intentó fusionarlos de nuevo (sobre ellos mismos que es como estaban) se produce naturalmente un conflicto.

Lo que jrandom realmente quería hacer era fusionar en su copia de trabajo los cambios entre la más reciente fusión del tronco con su estado actual. Usted puede hacer esto usando dos -j indicadores para actualizar, como debería recordar en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩, siempre que sepa que revisión corresponde con cada indicador. Afortunadamente qsmith hizó una etiqueta exactamente en el último punto de fusión (hurra por planificar con antelación!), por lo que esto no será problema. Primero veamos como jrandom puede devolver su copia de trabajo un estado limpio, desde el que puede rehacer la fusión:

```
floss$ rm README.txt a-subdir/whatever.c
floss$ cvs -q update
cvs update: warning: README.txt was lost
U README.txt
cvs update: warning: a-subdir/whatever.c was lost
U a-subdir/whatever.c
floss$
```

Ahora ella puede hacer la fusión, usando la etiqueta colocada convenientemente por qsmith.

```
floss$ cvs -q update -j Exotic_Greetings-1 -j Exotic_Greetings-branch
RCS file: /usr/local/newrepos/myproj/a-subdir/whatever.c,v
retrieving revision 1.3.2.1
retrieving revision 1.3.2.2
Merging differences between 1.3.2.1 and 1.3.2.2 into whatever.c
floss$ cvs -q update
M a-subdir/whatever.c
floss$
```

Mucho mejor. Los cambios de qsmith han sido incorporados a whatever.c; jrandom puede hacer una entrega y etiquetado:

```
floss$ cvs -q ci -m "merged again from Exotic_Greetings (1)"
```

```
Checking in a-subdir/whatever.c;
/usr/local/newrepos/myproj/a-subdir/whatever.c,v  <--  whatever.c
new revision: 1.6; previous revision: 1.5
done
floss$ cvs -q tag merged-Exotic_Greetings-1
T README.txt
T foo.gif
T hello.c
T a-subdir/whatever.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
floss$
```

Incluso si qsmith hubiese olvidado etiquetar en el punto de fusión, las esperanzas no estaría perdidas. Si jrandom supiese aproximadamente cuando hizo qsmith su primera entrega ella podría tratar de filtrar por la fecha:

```
floss$ cvs update -j Exotic_Greetings-branch:3pm -j Exotic_Greetings_branch
```

Aunque útil como último recurso, filtrar por fecha no es tan bueno porque selecciona los cambios basandose en los recuerdos de la gnete en vez de en designaciones que dependan del desarrollador. Si el primer conjunto de cambios fusionados de qsmith hubiera ocurrido en varias entregas en vez de sólo una jrandom pudiera equivocadamente elegir una fecha u hora que tomara algunos de los cambios, pero no todos.

No es necesario que cada punto etiquetado en los cambios de qsmith sea enviado al repositorio un una simple entrega. Ocurrió así casualmente en el ejemplo. En la vida real, qsmith pudo haber hecho varias entregas entre cada etiquetado. Él puede trabajar de forma aislada en su derivación tanto como quiera. La razón de las etiquetas es registar sucesivos puntos en la derivación donde considere que los cambios deban ser fusionados con la rama principal. Siempre que jrandom fusione usando dos indicadores -j y sea cuidadoso al usar las etiquetas de ramificación de qsmith en el orden apropiado y una sóla vez por cada un la rama principal padecer el problema de la doble fusión.

Podrían ocurrir conflictos, pero éstos serían de la inevitable clase que requiere resolución humana; situaciones en las que tanto el tronco como la derivación realizan cambios en la misma área de código.

La Aproximación de la Cola de Milano – Fusionar dentro y fuera de la rama principal

Fusionar repetidamente de derivación a tronco es bueno para la gente del tronco ya que ven todos sus cambios y los de la derivación. Sin embargo el desarrollador de la derivación no obtiene nunca los cambios producidos en el tronco.

Para permitir esto el desarrollador de la derivación debe realizar un paso extra cuando tenga ganas de hacer una fusión de los cambios más recientes del tronco y resolver los inevitables conflictos que surgan:

```
paste$ cvs update -j CABEZA
```

La etiqueta especial reservada `CABEZA` señala el estado actual del tronco. La orden anterior fusiona en el tronco los cambios entre la raiz de la actual derivación (`Exotic_Greetings-branch`) y la revisión más alta de cada fichero del tronco. Por

supuesto qsmith tiene que etiquetar de nuevo después de hacer esto para que los desarrolladores del tronco eviten accidentalmente fusionar sus propios cambios cuando intenten conseguir los de qsmith.

De la misma manera el desarrollador de la derivación puede usar las etiquetas de fusión del tronco como límites, permitinedo a la derivación fusionar exactamente aquellos cambios entre la última fusión y el estado actual del tronco (de la misma manera que el tronco fusiona). Por ejemplo, supongamos que jrandom ha hecho algunos cambios a hello.c después de fusionar la derivación:

```
floss$ emacs hello.c
 ...
floss$ cvs ci -m "clarify algorithm" hello.c
Checking in hello.c;
/usr/local/newrepos/myproj/hello.c,v  <--  hello.c
new revision: 1.22; previous revision: 1.21
done
floss$
```

Después puede qsmith fusionar esos cambios en el tronco, entregar, y, por supuesto, etiquetar.

```
paste$ cvs -q update -j merged-Exotic_Greetings-1 -j HEAD
RCS file: /usr/local/newrepos/myproj/hello.c,v
retrieving revision 1.21
retrieving revision 1.22
Merging differences between 1.21 and 1.22 into hello.c
paste$ cvs -q update
M hello.c
paste$ cvs -q ci -m "merged trunk, from merged-Exotic_Greetings-1 to HEAD"
Checking in hello.c;
/usr/local/newrepos/myproj/hello.c,v  <--  hello.c
new revision: 1.21.2.1; previous revision: 1.21
done
paste$ cvs -q tag merged-merged-Exotic_Greetings-1
T README.txt
T foo.gif
T hello.c
T a-subdir/whatever.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
paste$
```

Observe que a jrandom no etiquetó después de entregar los cambios a hello.c pero si qsmith. El principio utilizado aquí es que aunque usted no necesita etiquetar después de hacer cualquier pequeno cambio si debería hacerlo después de fusionar o entregar su línea de desarrollo a un estado que permita la fusión. De este modo otras personas, quizá en otras derivaciones tienen un punto de referencia en el cual basarse para sus propias fusiones.

La Aproximación del Pez Volador – Una forma más simple de hacerlo

Hay una variante más simple, aunque un poco más limitada, que la anterior. En ella, los desarrolladores de la derivación se congelan o paran mientras el tronco se fusiona, y después los desarrolladores del tronco crean una nueva derivación, que reemplaza la anterior. Los desarrolladores de la antigua derivación cambian a esa nueva derivación y siguen trabajando. Este ciclo sigue hasta uqe no hay más necesidad de desarrollo en la derivación. Funciona así (supondremos que jrandom@floss está en el tronco y qsmith@paste esté en la derivación como hasta ahora):

```
floss$ cvs tag -b BRANCH-1
paste$ cvs checkout -r BRANCH-1 myproj
```

Tronco y derivación empiezan a funcionar y pasado un tiempo los desarrolladores deciden que tienen que fusionar la derivación en el tronco:

```
paste$ cvs ci -m "committing all uncommitted changes"
floss$ cvs update -j BRANCH-1
```

Todos los cambios de la rama se fusionan y los desarrolladores paran su trabajo mientras que los desarrolladores del tronco resuelven cualquier conflicto, entregan, etiquetan y crean una nueva derivación:

```
floss$ cvs ci -m "merged from BRANCH-1"
floss$ cvs tag merged-from-BRANCH-1
floss$ cvs tag -b BRANCH-2
```

Ahora los desarrolladores de la derivación cambian sus copias de trabajo a la nueva derivación sabiendo que no perderán ningún cambio no entregado ya que estaban actualizados cuando la fusión ocurrió ya la nueva derivación viene de un tronco que incorpora los cambios de la nueva derivación:

```
paste$ cvs update -r BRANCH-2
```

Y el ciclo continúa del mismo modo indefinidamente; sustituya BRANCH-2 por BRANCH-1 y BRANCH-3 por BRANCH-2.

Llamo a esta técnica *Pez Volador* porque la derivación está constantemente emergiendo del tronco, viajando una corta distancia y uniéndose a él después. Las ventajas de esta aproximación son que es simple (el tronco siempre fusiona todos los cambios para una derivación dada) y los desarrolladores no tinene que resolver conflictos (cada vez trabajan con una nueva y limpia derivación). Las desventajas son que la gente de la derivación debe esperar sin poder hacer nada hasta que se haya fusionado el tronco (que puede durar un tiempo arbitrario dependiendo de cómo haya que resolver los conflictos). Otra pequeña desventaja es habrá muchas derivaciones sin usar en vez de etiquetas no_derivación. Si no le importa tener millones de pequeñas y obsoletas derivaciones y puede anticipar claramente fusiones libres de problemas el Pez Volador pueda ser el modo más fácil en términos mentales.

Derivaciones y expansión de palabras – Enemigos naturales

Si sus ficheros contienen palabras de expansión en RCS que se sustituyen de forma distinta en la derivación y en el tronco casi seguro que tendrá conflictos en las fusiones. Incluso si nada cambia las palabras de expansión se sobrepondrán y sus sustituciones no se podrán hacer. Por ejemplo; if README.txt contiene esto en el tronco

```
$Revision: 1.4 $
```

y esto en la derivación

```
$Revision: 1.4 $
```

entonces cuando sea realizada la fusión obtendrá el siguiente conflicto:

```
floss$ cvs update -j Exotic_Greetings-branch
RCS file: /usr/local/newrepos/myproj/README.txt,v
retrieving revision 1.14
retrieving revision 1.14.2.1
Merging differences between 1.14 and 1.14.2.1 into README.txt
rcsmerge: warning: conflicts during merge
floss$ cat README.txt
 ...
<<<<<<< README.txt
key $Revision: 1.4 $
=======
key $Revision: 1.4 $
>>>>>>> 1.14.2.1
 ...
floss$
```

Para evitar esto, usted puede temporalmente desabilitar la expansión de palabras pasando la opción -kk (No sé que significa; "kill keywords" (mata palabras) quizá?) cuando haga la fusión:

```
floss$ cvs update -kk -j Exotic_Greetings-branch
RCS file: /usr/local/newrepos/myproj/README.txt,v
retrieving revision 1.14
retrieving revision 1.14.2.1
Merging differences between 1.14 and 1.14.2.1 into README.txt
floss$ cat README.txt
 ...
$Revision: 1.4 $
 ...
floss$
```

Tiene que tener cuidado con una cosa; si usa -kk, invalida cualquier otro modo de expansión de palabras que pueda haber establecido para ese fichero. Específicamente esto es un problema para los ficheros binarios que normalmente usan -kb (que suprime todas las palabras de expansión y conversiones de fin de línea). Por lo tanto si tiene ficheros binarios en una derivación no use -kk. Trate los conflictos manualmente.

Vigilando fuentes de terceras partes (Derivaciones comerciales)

De vez en cuando un sitio pudiera hacer un cambio local al código de un programa obtenido del exterior. Si la fuente exterior no incorpora los cambios locales (y habría muchas razones legítimas para no hacerlo), el sitio tiene que mantener sus cambios en cada actualización del software.

CVS puede ayudar en esta tarea a través de una característica conocida como *derivaciones comerciales*. De hecho, derivaciones comerciales está detrás de los ahora misteriosos

dos finales argumentos de la orden `cvs import`; la etiqueta comercial y la de entrega final que vimos en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩.

He aquí como funciona. La importación inicial is como cualquier otra importación de un proyecto en CVS (excepto que tendrá que elegir la etiqueta comercial con un poco de cuidado):

```
floss$ pwd
/home/jrandom/theirproj-1.0
floss$ cvs import -m "Import of TheirProj 1.0" theirproj Them THEIRPROJ_1_0
N theirproj/INSTALL
N theirproj/README
N theirproj/src/main.c
N theirproj/src/parse.c
N theirproj/src/digest.c
N theirproj/doc/random.c
N theirproj/doc/manual.txt

No conflicts created by this import

floss$
```

Después debe obtener una copia de trabajo, hacer sus modificaciones locales y entregar:

```
floss$ cvs -q co theirproj
U theirproj/INSTALL
U theirproj/README
U theirproj/doc/manual.txt
U theirproj/doc/random.c
U theirproj/src/digest.c
U theirproj/src/main.c
U theirproj/src/parse.c
floss$ cd theirproj
floss$ emacs src/main.c src/digest.c
 ...
floss$ cvs -q update
M src/digest.c
M src/main.c
floss$ cvs -q ci -m "changed digestion algorithm; added comment to main"
Checking in src/digest.c;
/usr/local/newrepos/theirproj/src/digest.c,v  <--  digest.c
new revision: 1.2; previous revision: 1.1
done
Checking in src/main.c;
/usr/local/newrepos/theirproj/src/main.c,v  <--  main.c
new revision: 1.2; previous revision: 1.1
done
floss$
```

Un año más tarde la siguiente versión del programa llega de Ellos, S.A., y ustede debe incorporar sus cambios locales a ella. Los cambios de ellos y los suyos se sobreponen ligeramente. Ellos han añadido un nuevo fichero, modificado un par de ficheros que usted no tocó y otros dos que usted si modificó.

Primero tiene que hacer otra importación para obtener las nuevas fuentes. Casi todo estaba como en la importación inicial; usted está importando el mismo proyecto del repositorio y de la misma derivación comercial. La única diferencia es en la etiqueta de entrega final:

```
floss$ pwd
/home/jrandom/theirproj-2.0
floss$ cvs -q import -m "Import of TheirProj 2.0" theirproj Them THEIRPROJ_2_0
U theirproj/INSTALL
N theirproj/TODO
U theirproj/README
cvs import: Importing /usr/local/newrepos/theirproj/src
C theirproj/src/main.c
U theirproj/src/parse.c
C theirproj/src/digest.c
cvs import: Importing /usr/local/newrepos/theirproj/doc
U theirproj/doc/random.c
U theirproj/doc/manual.txt

2 conflicts created by this import.
Use the following command to help the merge:

       cvs checkout -jThem:yesterday -jThem theirproj

floss$
```

Dios mío; No hemos visto que CVS sea tan útil. Nos está diciendo que orden ejecutar para fusionar los cambios. Y casi está bien. En realidad el comando funciona (asumiendo que sustituye `yesterday` (ayer) por un intervalo de tiempo que incluya la primera primera importación pero no la segunda). Yo prefiero hacerlo mediante etiquetas de entrega final:

```
floss$ cvs checkout -j THEIRPROJ_1_0 -j THEIRPROJ_2_0 theirproj
cvs checkout: Updating theirproj
U theirproj/INSTALL
U theirproj/README
U theirproj/TODO
cvs checkout: Updating theirproj/doc
U theirproj/doc/manual.txt
U theirproj/doc/random.c
cvs checkout: Updating theirproj/src
U theirproj/src/digest.c
RCS file: /usr/local/newrepos/theirproj/src/digest.c,v
retrieving revision 1.1.1.1
retrieving revision 1.1.1.2
Merging differences between 1.1.1.1 and 1.1.1.2 into digest.c
rcsmerge: warning: conflicts during merge
U theirproj/src/main.c
RCS file: /usr/local/newrepos/theirproj/src/main.c,v
retrieving revision 1.1.1.1
retrieving revision 1.1.1.2
Merging differences between 1.1.1.1 and 1.1.1.2 into main.c
```

```
U theirproj/src/parse.c
floss$
```

Observe como la importación nos indica que hay dos conflictos pero la fusión parece ver sólo uno. Esto es porque la idea de conflicto en CVS es un poco diferente que en las otras ocasiones. Básicamente la importación informa de conflictos cuando usted y el distribuidor modifican un fichero entre la última importación y esta. Sin embargo, cuando se fusiona o actualiza la definición de conflicto es la usual, cambios que se sobreponen. Cambios que no se sobreponen son fusionados de la forma normal y el fichero se marca como modificado.

Un `diff` verifica que sólo uno de los ficheros tiene conflictos:

```
floss$ cvs -q update
C src/digest.c
M src/main.c
floss$ cvs diff -c
Index: src/digest.c
===================================================================
RCS file: /usr/local/newrepos/theirproj/src/digest.c,v
retrieving revision 1.2
diff -c -r1.2 digest.c
*** src/digest.c        1999/07/26 08:02:18     1.2
-- src/digest.c        1999/07/26 08:16:15
***************
*** 3,7 ****
-- 3,11 ----
 void
 digest ()
 {
+ <<<<<<< digest.c
   printf ("gurgle, slorp\n");
+ =======
+   printf ("mild gurgle\n");
+ >>>>>>> 1.1.1.2
 }
Index: src/main.c
===================================================================
RCS file: /usr/local/newrepos/theirproj/src/main.c,v
retrieving revision 1.2
diff -c -r1.2 main.c
*** src/main.c  1999/07/26 08:02:18     1.2
-- src/main.c  1999/07/26 08:16:15
***************
*** 7,9 ****
-- 7,11 ----
 {
   printf ("Goodbye, world!\n");
 }
+
+ /* I, the vendor, added this comment for no good reason. */
floss$
```

A partir de aquí deberá resolver los conflictos como cualquier otra fusión:

```
floss$ emacs  src/digest.c  src/main.c
 ...
floss$ cvs -q update
M src/digest.c
M src/main.c
floss$ cvs diff src/digest.c
cvs diff src/digest.c
Index: src/digest.c
===================================================================
RCS file: /usr/local/newrepos/theirproj/src/digest.c,v
retrieving revision 1.2
diff -r1.2 digest.c
6c6
<   printf ("gurgle, slorp\n");
--
>   printf ("mild gurgle, slorp\n");
floss$
```

Entones entregue los cambios

```
floss$ cvs -q ci -m "Resolved conflicts with import of 2.0"
Checking in src/digest.c;
/usr/local/newrepos/theirproj/src/digest.c,v  <--  digest.c
new revision: 1.3; previous revision: 1.2
done
Checking in src/main.c;
/usr/local/newrepos/theirproj/src/main.c,v  <--  main.c
new revision: 1.3; previous revision: 1.2
done
floss$
```

y espere la próxima versión del distribuidor. (Por supuesto tendrá que comprobar que sus antiguas modificaciones todavía funcionan).

Exportar para distribución pública

CVS es un buen mecanismo de distribución para desarrolladores, pero la mayoría de usuarios obtendrán el software a través de un paquete descargable. Este paquete normalmente no es una copia de trabajo de CVS; es un árbol de código que puede ser fácilmente configurado y compilado en el sistema del usuario.

Sin embargo, CVS ofrece un mecanismo que ayuda a crear ese paquete, la orden `cvs export` (Exportar). *Exportar* un proyecto es como obtener una copia de trabajo del proyecto, excepto que se obtiene el directorio completo del proyecto *sin* los subdirectorios administrativos. O sea, que no obtiene una copia de trabajo sino el código fuente completo que no sabe nado sobre dónde vino o que versiones de CVS tienen sus ficheros. Así la copia exportada es como lo que el público ve cuando descarga y desempaqueta un distribución. Asumiendo que el proyecto está organizado para que sea directamente compilable desde la copia de trabajo (y así es como debería estar), entonces todavía será compilable en la copia exportada.

La orden `export` funciona igual que `checkout`, excepto que requiere una etiqueta o fecha. Por ejemplo, aquí hemos etiquetado el proyecto con un nombre para el lanzamiento final y hemos exportado basándonos en eso:

```
floss$ pwd
/home/jrandom/myproj
floss$ cvs -q tag R_1_0
T README.txt
T hello.c
T a-subdir/whatever.c
T a-subdir/subsubdir/fish.c
T b-subdir/random.c
floss$ cd ..
floss$ cvs -d /usr/local/newrepos -q export -r R_1_0 -d myproj-1.0 myproj
U myproj-1.0/README.txt
U myproj-1.0/hello.c
U myproj-1.0/a-subdir/whatever.c
U myproj-1.0/a-subdir/subsubdir/fish.c
U myproj-1.0/b-subdir/random.c
floss$ cd myproj-1.0
floss$ ls
README.txt  a-subdir  b-subdir  hello.c
```

Observe que como la `export` no es llamada desde una copia de trabajo ha sido necesario usar la opción global `-d` para decirle a CVS qué repositorio usar. En este ejemolo en particular, además, exportamos a un directorio explícitamente nombrado (`myproj-1.0`) en vez del directorio por defecto con el nombre del proyecto (`myproy`, porque ya había una copia con ese nombre presente. Esta situación no es infrecuente.

Después de crear la copia mediante export, como en el ejemplo anterior, lo que sigue es suficiente para completar la entrega final si el proyecto es sencillo:

```
floss$ tar cf myproj-1.0.tar myproj-1.0
floss$ gzip --best myproj-1.0.tar
floss$ ls
myproj/   myproj-1.0/   myproj-1.0.tar.gz
floss$ rm -rf myproj-1.0
floss$ mv myproj-1.0.tar.gz /home/ftp/pub/myproj/
```

Ejecutar todas estas órdenes a mano es raro. Lo normal es que `cvs export` sea llamada desde una rutina que maneje todos los aspectos de la entrega final y el proceso de empaquetado. Debido a que hay varias entregas de prueba antes del lanzamiento final es deseable que los procedimientos para crear un paquete se automatizen.

El humilde gurú

Si usted ha leido y comprendido e incluso mejor, ha experimentado con todo lo de este capítulo, puede asegurar que no le quedan muchas más cosas en CVS por aprender; al menos hasta que alguien añada una nueva e importante característica a CVS. Todo lo que necesita saber para usar CVS en un gran proyecto ha sido presentado.

Antes de que esto se le suba a la cabeza permítame insistir en la sugerencia, hecha ya en el capítulo 4, de que se suscriba a la lista de correo `info-cvs@gnu.org`. A pesar

de una proporción de ruido común a la mayoría de listas de correo en Internet, lo bueno que le llega merece la pena esperarlo. He estado suscrito durante el tiempo que estuve escribiendo este capítulo y estaría sorprendido de la cantidad de importantes de detalles que aprendí sobre el comportamiento de CVS leyendo los correos de otras personas. Si va a usar CVS seriamente y especialmente si usted es un administrador de CVS en un grupo de desarrolladores se puede beneficiar del conocimiento compartido entre todos los demás usuarios serios que hay por ahí.

Problemas y Soluciones

Comenté en capítulos anteriores que CVS no es un software de "caja negra". Las cajas negras no le permiten ver lo que hay dentro; no le dan acceso interno que le permita arreglar (o estropear) cosas. La causa es que la caja negra normalmente no necesitará ser revisada por usted. Casi siempre, el software funcionará correctamente, por lo que los usuarios no necesitarán acceso interno. Pero cuando las cajas negras fallan, tienden a hacerlo completamente. Cualquier problema será un "exitazo", puesto que no hay muchas opciones para reparar.

CVS es más bien como una caja perfectamente transparente. Sus partes "móviles" están expuestas directamente al entorno, y fallos de ese entorno (permisos inesperados en ficheros, comandos interrumpidos, procesos en competencia, etc) pueden a veces influir en el mecanismo interno y producir fallos. Pero aunque CVS no siempre funciona a la perfección, raramente falla por completo. Tiene la ventaja de tener una "degradación gradual": el porcentaje de fallos es proporcional al número y severidad de problemas con el entorno. Si tenemos suficiente idea acerca de lo que CVS está tratando de hacer – y cómo lo quiere hacer – sabremos qué hacer cuando las cosas van mal.

Aunque no puedo listar todos los problemas que puede encontrar, he incluido algunos de los más habituales. Este capítulo está dividido en dos partes: la primera describe aquellas partes del entorno a las que CVS es más sensible (principalmente, permisos del repositorio y de la copia de trabajo del área administrativa), y la segunda describe algunos de los problemas que se encuentran con más frecuencia y sus soluciones. Observando cómo se gestionan esas situaciones, obtendremos una capacidad para acercarnos a la solución de otros problemas con CVS no descritos aquí.

Causas Usualmente Sospechosas

Como administradores de CVS (léase "médicos del CVS"), encontraremos que el 90 por ciento de los problemas de nuestros usuarios estarán causados por copias de trabajo inconsistentes, y el otro 90 por ciento por permisos incorrectos en el repositorio. No obstante, antes de investigar cualquier situación concreta, mostraremos una vista rápida de la copia de trabajo del área administrativa y revisaremos algunas cosas importantes acerca de los permisos en el repositorio.

La Copia de Trabajo del Área Administrativa

Ya vimos la estructura de la copia de trabajo en ⟨undefined⟩ [Una introduccion a CVS], page ⟨undefined⟩; en esta sección entraremos un poco más en detalle. Casi todos los detalles conciernen a los ficheros de los directorios administrativos bajo CVS/. Ya conocemos los ficheros Entries, Root y Repository, pero en el directorio CVS/ también puede haber otros ficheros, según las circunstancias. Describiré esos ficheros aquí, parcialmente para que no nos sorprenda encontrarlos, y también para que sepamos cómo corregir errores que eventualmente se produzcan en ellos.

'CVS/Entries.Log'

A veces, aparecerá un extraño fichero '`CVS/Entries.Log`'. El único propósito de este fichero es hacer de cache temporal de los cambios menores de CVS/Entries, hasta que haya suficientes cambios acumulados para modificar este fichero. CVS no es capaz de editar directamente el fichero Entries, por el contrario tiene que leerlo y sobreescribirlo por completo para cualquier cambio. Para evitar excesiva carga, CVS a veces guarda los cambios pequeños en Entries.log, hasta la próxima vez que necesite reescribir el fichero Entries.

El formato de Entries.log es como el de Entries, salvo que además incluye una letra al principio de cada línea. `A` significa que la línea será añadida al fichero Entries, y `R` significa que esa línea será borrada.

Casi siempre podemos ignorar el fichero Entries.log; es raro que un administrador tenga que entender la información que contiene. Sin embargo, si estamos depurando algún problema que nos pide examinar el fichero Entries, probablemente tengamos que echar un vistazo también a Entries.log.

'CVS/Entries.Backup'

El fichero CVS/Entries.backup es el que usa CVS para escribir un fichero Entries nuevo, antes de renombrarlo a '`Entries`' (similarmente al mecanismo de escribir en ficheros temporales RCS y luego renombrarlos convenientemente). Debido a que se renombra a Entries cuando está acabado, serán pocas las veces que veremos el fichero Entries.Backup; si lo vemos alguna vez se deberá probablemente a la interrupción de un proceso CVS en medio de alguna operación.

'CVS/Entries.Static'

Si existe el fichero CVS/Entries.Static, significa que el directorio completo no ha sido extraido del repositorio. (Cuando CVS sabe que un directorio está en un estado incompleto, él no añadirá ficheros adicionales a ese directorio.)

El fichero Entries.Static existe durante las operaciones de checkout y actualización, y se borra inmediatamente al completar la operación. Si vemos el fichero, significará que CVS fue interrumpido, y su existencia impide a CVS crear cualquier fichero nuevo en la copia de trabajo. (Ahora bien , ejecutando `cvs update -d` se soluciona el problema y se borra Entries.Static.)

La ausencia de Entries.Static no implica necesariamente que la copia de trabajo incluya todos los ficheros del proyecto. Tan pronto se crea un nuevo directorio en el repositorio del proyecto, y alguien actualiza su copia sin incluir el flag -d a la orden update, el nuevo directorio no se creará en la copia de trabajo. Localmente, CVS no sabe que existe el nuevo directorio, luego él mismo borrará el fichero Entries.Static cuando la actualización termine, aunque el nuevo directorio no se haya creado en la copia de trabajo.

'CVS/Tag'

Si existe el fichero CVS/Tag, nombrará a un tag asociado, en cierto sentido, con el directorio. Digo "en cierto sentido" ya que, como sabemos, CVS no mantiene información

sobre versiones de directorios y, hablando con precisión, no puede asociar tags a ellos. Los tags se asocian solo a ficheros normales o, más precisamente, a revisiones concretas de ficheros normales.

Sin embargo, si cada fichero de un directorio está en un tag concreto, CVS presupone que el directorio también está en ese tag. Por ejemplo, si íbamos a extraer de CVS una copia de trabajo de una rama concreta:

```
floss$ cvs co -r Bugfix_Rama_1
```

y luego insertamos un fichero en ella, querremos que la versión inicial del fichero esté en esa rama también. Por razones similares, CVS también necesita conocer si el directorio tiene un tag de que no es rama (non-branch) o la fecha puesta en él.

Los ficheros de tags contienen una línea. El primer carácter de la línea es un código de una letra que indica qué clase de tag es, y el resto es el nombre del tag. Actualmente, CVS solo utiliza las siguientes tres letras como código:

T – Tag de rama

N – Tag de que no es rama (regular tag)

D – Fecha "pegada", que se utiliza si un comando como

```
floss$ cvs checkout -D 1999-05-15 myproj
```

o

```
floss$ cvs update -D 1999-05-15 myproj
```

se ejecuta.

(Si vemos cualquier otro código de una letra, simplemente nos indicará que CVS ha añadido un nuevo tipo de tag posteriormente a la escritura de este libro.)

El fichero Tag no debe ser borrado manualmente; en su lugar use `cvs update -A`.

Rarezas

Hay otros ficheros que ocasionalmente se encontrarán en un directorio CVS/:

CVS/Checkin.prog, CVS/Update.prog

CVS/Notify, CVS/Notify.tmp

CVS/Base/, CVS/Baserev, CVS/Baserev.tmp

CVS/Template

Estos ficheros no son, normalmente, causa de problemas, por lo que simplemente los vamos a listar (véase ⟨undefined⟩ [Referencia de CVS], page ⟨undefined⟩ para conocer su significado completo).

Portabilidad y extensiones futuras.

Conforme se añadan nuevas características a CVS, podrán aparecer nuevos ficheros (no listados aquí) en las áreas administrativas. Conforme dichos ficheros sean añadidos, probablemente serán documentados en el manual de Cederqvist, en la sección *Working Directory Storage*. También podemos mirar en el código fuente, en src/cvs.h, si preferimos aprender a partir de las fuentes.

Finalmente, observemos que todos los ficheros CVS/* – actuales y futuros – siguen las convenciones de final de línea apropiadas al sistema en uso (por ejemplo, LF para Unix o CR/LF para Windows). Esto significa que si llevamos una copia de trabajo desde una plataforma a otra, puede suceder que CVS no pueda manejarla (además, podemos tener otros problemas, debido a que los ficheros controlados por el sistema de versiones pueden tener ellos mismos un fin de línea inapropiado).

Permisos del Repositorio

CVS no necesita ningún esquema específico de permisos – puede manejar una amplia variedad de esquemas. Sin embargo, para evitar situaciones confusas, se debe configurar el repositorio siguiendo como mínimo los siguientes criterios:

Si un usuario quiere algún tipo de acceso – incluso acceso solo-lectura – a un determinado directorio del repositorio, normalmente necesitará permisos de escritura a nivel de sistema sobre ese directorio. Esto es necesario ya que CVS crea ficheros cerrojo temporales en el repositorio para asegurarse la consistencia de los datos. Aun en operaciones de solo lectura (como la extracción o actualización de una copia de trabajo), se crearán cerrojos, para asegurar que los datos permanecen constantes durante la operación.

Como se indica en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩, podemos salvar este requisito ajustando el parámetro LockDir del fichero CVSROOT/config. Por ejemplo:

```
LockDir=/usr/local/cvslocks
```

Por supuesto, habrá que asegurarse de que todos los usuarios de CVS pueden escribir sobre /usr/local/cvslocks. De otro modo, el directorio será el del repositorio; si somos muy estrictos con la seguridad, deberíamos cambiar este directorio por otro.

Asegurémosnos de que el fichero CVSROOT/history es escribible por todo el mundo, ya que si este fichero existe, casi todas las operaciones de CVS intentarán concatenar su histórico en este fichero, y si no lo pueden hacer terminarán con error.

Por desgracia (e inexplicablemente), el fichero de histórico no se crea escribible por todo el mundo cuando se crea un repositorio nuevo con cvs init. Al menos con la versión actual de CVS, debemos cambiar los permisos de manera explícita una vez creado el repositorio (o simplemente borrarlo, si no queremos que se almacenen los históricos).

(Este problema puede terminar pronto – acabo de enviar un parche a los mantenedores de CVS para que se cree el histórico con los permisos apropiados cuando se crea. Luego si usamos una versión de CVS posterior a septiembre de 1999, probablemente el problema habrá desaparecido.)

Por razones de seguridad, hay que asegurarse de que los usuarios de CVS no tienen acceso de escritura a nivel de Unix al directorio CVSROOT. Si alguien tiene acceso de inserción en CVSROOT, podría editar cualquier fichero disparador de su elección (commitinfo, loginfo, etc), invocando cualquier programa que desee. En general, el acceso a CVSROOT por parte de un usuario de CVS implica la posibilidad de ejecutar cualquier comando del sistema.

Trucos Habituales

Este capítulo está organizado como una serie de preguntas y respuestas, de manera similar a una FAQ (Preguntas Frecuentes) de Internet. Todas ellas se basan en la experiencia habitual con CVS. Pero antes de mostrar casos más individuales, tomemos unos minutos para considerar los problemas de CVS desde un punto de vista más general.

El primer paso en la resolución de un problema con CVS es determinar si sucede en una copia de trabajo o es un problema del repositorio. La mejor técnica para hacerlo, es ver si el problema se repite en copias de trabajo distintas del mismo repositorio. Si sucede así, será probablemente un problema del repositorio; en otro caso estará limitado a la copia local.

Los problemas con las copias de trabajo suelen encontrarse más frecuentemente, simplemente porque son más numerosas que los repositorios y no porque sean más "inestables". Aunque con algo de paciencia se pueden resolver la mayoría de los problemas, muchas veces la solución más sencilla y eficiente será borrar y volver a crear la copia de trabajo.

Por supuesto, si crear la copia de trabajo resulta tedioso, o hay muchas cosas pendientes de enviar al repositorio como para permitirse el lujo de borrarlo, o simplemente somos de los que queremos averiguar el por qué de las cosas, siempre podemos intentarlo. En primer lugar, normalmente miraremos los subdirectorios CVS/, comprobando los ficheros y sus permisos. A veces, los permisos se hacen misteriosamente de solo lectura o simplemente sin permisos de lectura. Sospechamos que es debido a que los usuarios se equivocan con alguna orden Unix relacionada y cambian los permisos sin saberlo.

Los problemas en el repositorio se suelen producir por permisos incorrectos en ficheros y directorios. Si sospechamos que el problema pueda deberse a esto, primero miremos cuál es el UID efectivo en el repositorio de la persona que origina el problema. Con usuarios locales y remotos, suele ser un problema con el usuario que se especificó durante la creación de la copia de trabajo. Si se usa el método pserver con alias de usuario (véase la sección ⟨undefined⟩ [Acceso anonimo], page ⟨undefined⟩ en el capítulo ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩), el ID de usuario efectivo estará en el fichero CVSROOT/passwd. Si no se ve esto a tiempo puede provocarnos una gran pérdida de tiempo buscando la solución al problema en otros sitios.

Y ahora veamos problemas más concretos...

Algunos Problemas de la Vida Real (con Soluciones)

Las siguientes son situaciones que se me han dado en mi experiencia como administrador de CVS (más algunas cosas que no son problemas realmente, simplemente cuestiones que he oido por ahí y creo interesante contar aquí). La lista pretende ser bastante completa y puede repetir cosas que ya hayamos visto en capítulos anteriores.

Las situaciones se listan de acuerdo a la frecuencia con la que se suelen dar, poniendo en primer lugar las más habituales.

CVS dice que está esperando un cerrojo; qué significa esto?

Si vemos un mensaje como este:

```
cvs update: [22:58:26] waiting for qsmith's lock in /usr/local/newrepos/myproj
```

significa que estamos intentando acceder a un subdirectorio del repositorio que está bloqueado por otro proceso CVS en este momento. Un proceso está corriendo en ese subdirectorio, luego puede hacerlo inconsistente a otros procesos CVS que quieran acceder a él.

Sin embargo, si el mensaje de espera persiste mucho tiempo, probablemente indique que un proceso CVS ha fallado en su limpieza final, por alguna razón. Puede pasar cuando CVS muere de pronto e inesperadamente, por una caída de la máquina del repositorio, por ejemplo.

La solución es borrar los ficheros de cerrojo a mano del subdirectorio del repositorio en cuestión. Entremos en ese lugar y busquemos ficheros con el nombre '`#cvs.lock`' o que empiecen por '`#cvs.wfl`' o '`#cvs.rfl`'. Comparemos las fechas de los ficheros con los instantes de inicio de cualquier proceso CVS actual. Si los ficheros no han podido ser creados por esos procesos (son más antiguos), podemos borrarlos tranquilamente. Los procesos CVS en espera se darán cuenta del borrado (normalmente cada 30 segundos lo revisan) y terminarán su operación.

Véase el nodo *Locks* en el manual de Cederqvist para más detalle.

CVS dice que un fichero ha fallado la comprobación Up-To-Date (actualizado); qué hago?

Evitemos el pánico – solo significa que el fichero ha cambiado en el repositorio desde la última vez que nos lo bajamos o actualizamos.

Ejecutemos `cvs update` en el fichero para mezclar los cambios del repositorio con los que hayamos hecho nosotros. Si los cambios recibidos entran en conflicto con los nuestros, edítese el fichero para resolver los conflictos. A continuación intentemos de nuevo enviar los cambios al repositorio – tendrá exito, a menos que otra persona haya vuelto a actualizar el fichero durante su trabajo.

El método de acceso pserver no funciona

La causa más común, pero no precisamente obvia, es que hayamos olvidado la opción `--allow-root` en la configuración de inetd.

Recordemos este ejemplo de línea de /etc/inetd.conf:

```
cvspserver stream tcp nowait root /usr/local/bin/cvs cvs \
          --allow-root=/usr/local/newrepos pserver
```

(En el fichero real será una sola línea, sin barra invertida.)

La parte `--allow-root=/usr/local/newrepos` es una medida de seguridad, para asegurarnos de que la gente no pueda usar CVS para obtener acceso pserver a repositorios que se supone no son servidos remotamente. Cualquier repositorio que se desee accesible por pserver, debe estar mencionado en la opción `--allow-root`. Podemos tener todas las opciones de este tipo que deseemos, para dar acceso a todos los repositorios que hagan falta (mientras no llenemos la longitud máxima de línea de la configuración del inetd).

Véase el capítulo ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ para más detalle sobre la configuración del servidor autentificado con contraseña.

El método pserver SIGUE sin funcionar

De acuerdo, si el problema no es la ausencia de una opción `--allow-root`, veamos otras posibles causas:

El usuario no tiene una entrada en el fichero CVSROOT/passwd, y el fichero CVSROOT/config tiene la opción SystemAuth=no, por lo que CVS no buscará el usuario en el fichero de usuarios del sistema (o bien SystemAuth=yes pero no existe ese usuario en el sistema).

El usuario tiene una entrada en el fichero CVSROOT/passwd, pero no hay usuario con ese nombre en el sistema, por lo que no se puede mapear a un usuario válido.

La contraseña es incorrecta (aunque como CVS suele informar de esto convenientemente, casi seguro que nuestro problema no será éste).

Todo está correcto en los ficheros de claves y en /etc/inetd.conf, pero se nos olvidó una entrada como ésta en /etc/services:

```
cvspserver      2401/tcp
```

por lo que inetd no es capaz de saber qué puerto es cvspserver.

Mis envíos (commits) parecen tener lugar a trozos y no atómicamente

Esto es porque CVS hace los envíos a trozos, y no atómicamente. :-)

Más específicamente, las operaciones de CVS tienen lugar directorio a directorio. Cuando hacemos un commit (o update o cualquier cosa) afectando a varios directorio, CVS bloquea cada directorio mientras hace la operación en él, desbloqueándolo antes de pasar al siguiente.

Para proyectos pequeños o medianos, raramente será esto un problema, ni notaremos que la operación no es atómica. Sin embargo, en proyectos grandes, se pueden dar escenarios como el siguiente (imaginemos que el proyecto tiene al menos dos directorios A y B, con muchos ficheros):

1. El usuario pperez inicia un envío (commit), afectando a ficheros de ambos directorios. CVS envía los ficheros de B en primer lugar (porque el usuario lo especificó en ese orden).
2. El usuario jsuerte inicia una actualización (update). Por alguna razón, supongamos que ésta se inicia copiando el directorio A (CVS no garantiza ningún orden por su cuenta). Obsérvse que no hay bloqueo aun porque pperez aun no está activo en A.
3. Ahora, el envío de pperez finaliza B, se va a A y finaliza A.
4. Finalmente, la actualización de jsuerte se va a B y finaliza.

Claramente, cuando todo acaba, la copia de trabajo de jsuerte refleja los cambios de pperez en B pero no en A. Aunque pperez intentase hacerlo atómicamente, no hay forma. Ahora la copia de jsuerte está en un estado que desconoce pperez.

La solución, por supuesto, es que jsuerte haga de nuevo el cvs update.

El fallo de no permitir transacciones atómicas es considerado ampliamente como un error de CVS. La única razón por la que los cerrojos no se establecen en la raíz del repositorio es porque esto resultaría inaceptable para grandes proyectos con múltiples desarrolladores.

Para mitigar este problema, en CVS se escogió bloquear a nivel de cada directorio, reduciendo así la contención. Alguna vez alguien podría modificar CVS para acelerar sus operaciones, de manera que se mejore esta situación.

Para más información, véase el nodo *Concurrency* del manual de Cederqvist.

CVS ignora los permisos que pongo; por qué lo hace?

En general, CVS no realiza un muy buen trabajo para preservar los permisos de los ficheros. Cuando importamos un proyecto y luego lo extraemos, no hay garantía de que en la copia de trabajo obtenida los ficheros tengan los mismos permisos que cuando fueron importados. Más bien, lo que sucede es que los ficheros de la copia de trabajo se crean con el esquema de permisos estándar que tengamos en nuestra cuenta de usuario.

Sin embargo, hay al menos una excepción. Si queremos almacenar scripts de shell ejecutables en el proyecto, podemos mantenerlos ejecutables en todas las copias de trabajo sin más que hacer ejecutable el fichero del repositorio:

```
floss$ ls -l /usr/local/mirepo/unproyecto
total 6
-r--r--r--   1 jsuerte  users         630 Aug 17 01:10 README.txt,v
-r-xr-xr-x   1 jsuerte  users        1041 Aug 17 01:10 scrub.pl,v*
-r--r--r--   1 jsuerte  users         750 Aug 17 01:10 hola.c,v
```

Nótese que aunque el fichero es ejecutable, se mantiene en solo-lectura, como debe ser en todos los ficheros de un repositorio (recordar que CVS trabaja haciendo los cambios sobre una copia temporal del fichero RCS, que luego reemplaza al original).

Cuando importamos o añadimos un fichero ejecutable, CVS preserva los bits de ejecución, de manera que si los permisos iniciales son correctos, en general no habrá que preocuparse más. Sin embargo, si accidentalmente añadimos el fichero antes de hacerlo ejecutable, debemos ir al repositorio y cambiar los bits a mano sobre el fichero RCS.

Los permisos del repositorio siempre predominan. Si el fichero no es ejecutable en el repositorio pero sí lo es en la copia de trabajo, cuando hagamos una actualización seguirá como esté en el repositorio. Cuando los permisos de los ficheros cambian misteriosamente puede ser frustrante. Si esto sucede, comprobar primero los permisos en el repositorio y ver si podemos resolverlo ajustando los permisos sobre los ficheros RCS.

Recientemente se añadió a CVS una característica denominada `PreservePermissions` que puede aliviar alguno de estos problemas. Sin embargo, usando esta característica pueden producirse otros resultados inesperados (por lo que no recomiendo usarla siempre). Nos debemos asegurar de leer antes los nodos *config* y *Special Files* del manual de Cederqvist antes de incluir `PreservePermissions=yes` en CVSROOT/config.

El CVS de Windows dice que no puede encontrar mi fichero .cvspass por qué?

Para conexiones pserver, el CVS del lado cliente intenta encontrar el fichero .cvspass en el directorio principal de la cuenta (HOME). Las máquinas con Windows no tienen un directorio "home" natural, por lo que CVS consulta la variable `%HOME%`. Sin embargo, hay que ser cuidadosos con esta variable. Esto funcionará:

```
set HOME=C:
```

Pero esto no:

```
set HOME=C:\
```

Esta barra extra es suficiente para confundir a CVS y será incapaz de abrir el fichero '`C:\\.cvspass`'.

La más rápida y segura solución, pues, es poner

```
set HOME=C:
```

en el fichero autoexec.bat y reiniciar. El pserver de CVS debe funcionar ahora correctamente.

Mi copia de trabajo está en diferentes ramas una ayuda?

Hablamos de diferentes subdirectorios de la copia de trabajo en diferentes ramas? Probablemente hemos ejecutado updates con la opción -r, pero en lugares distintos de la raíz de la copia de trabajo.

No hay problema. Si queremos volver a lo correcto, ejecutemos esto

```
cvs update -r HEAD
```

o esto

```
cvs update -A
```

desde el directorio raíz. O, si lo que queremos es poner la copia de trabajo a una de las ramas, hacer esto:

```
cvs update -r Nombre_rama
```

No hay problema por tener uno o dos subdirectorios de la copia de trabajo pertenecientes a diferentes ramas, si lo que queremos es hacer algún trabajo en esa rama temporalmente solo en esos ficheros. Sin embargo, será una buena idea normalmento volver cuando acabemos – la vida será mucho menos confusa cuando toda nuestra copia de trabajo pertenezca a la misma línea de desarrollo.

Cuando hago export -d a veces pierdo commits recientes

Esto se debe a una diferencia entre el reloj de la máquina del repositorio y el local. Podemos resolverlo ajustando uno o ambos relojes, o especificando una fecha distinta con la opción -D. Es perfectamente aceptable el especificar una fecha del futuro (tal como -D tomorrow), si esto puede compensar la diferencia de tiempos.

Obtengo un error de val-tags; qué hago?

Si obtenemos un error como este:

```
cvs [export aborted]: cannot write /usr/local/myproj/CVSROOT/val-tags: \
   Operation not permitted
```

significa que el CVS del usuario está corriendo y no tiene permiso para escribir el fichero CVSROOT/val-tags. Este fichero almacena nombres de tags, para que CVS tenga una manera rápida de determinar qué tags son válidos. Desafortunadamente, CVS a veces modifica este fichero en operaciones que deberían ser solo-lectura respecto del repositorio, como una simple extracción (check-out) del proyecto.

Esto es un error de CVS y debe haberse corregido mientras leemos esto. Hasta entonces, la solución es hacer que el fichero val-tags sea escribible por todo el mundo o, si esto falla, borrarlo o poner de propietario al usuario que está intentando operar. (Podríamos pensar que cambiar los permisos es suficiente, pero a veces he tenido que tambiar también al propietario.)

Tengo problemas con los tags adhesivos; cómo evitarlos?

Algunas operaciones del CVS hacen que la copia de trabajo tengan un *tag adhesivo*, que es un tag que corresponde a cada revisión de cada fichero (en el caso de una rama, el tag adhesivo se aplica a cualquier fichero que se añada a la copia de trabajo). Obtendremos un área de trabajo con tags adhesivos cuando extraigamos o actualicemos por tag o por fecha, por ejemplo:

```
floss$ cvs update -r Nombre_Tag
```

o

```
floss$ cvs checkout -D '1999-08-16'
```

Si se usa una fecha o un nombre de tag que no sea rama, la copia de trabajo será una foto congelada de ese momento en el histórico del proyecto – por lo que naturalmente no podremos enviar cambios de ninguna clase desde él.

Para eliminar un tag adhesivo actualizaremos con el flag -A

```
floss$ cvs update -A
```

que limpia todos los tags adhesivos y actualiza cada fichero a su revisión más reciente.

Las extracciones/actualizaciones terminan con el error 'cannot expand modules'

Esto es un ejemplo de error fatal de CVS; probablemente alguien lo intentará corregir pronto, pero mientras tanto nos molestará. El error es similar al siguiente:

```
floss$ cvs co -d bwf-misc user-space/bwf/writings/misc
cvs server: cannot find module `user-space/bwf/writings/misc' - ignored
cvs [checkout aborted]: cannot expand modules
```

CVS aparenta estar diciendo que hay algo mal en el fichero CVSROOT/modules. Sin embargo, lo que realmente ocurre es un problema de permisos del repositorio. El directorio que estamos intentando extraer no es legible, o uno de sus ancestros no lo es. En este caso, era un ancestro:

```
floss$ ls -ld /usr/local/cvs/user-space/bwf

drwx------  19 bwf      users        1024 Aug 17 01:24 bwf/
```

Como vemos no hay que preocuparse demasiado – es simplemente otro problema de permisos.

No puedo desactivar los watches

Probablemente habremos ejecutado

```
floss$ cvs watch remove
```

en todos los ficheros, pero se nos olvidó hacer esto otro:

```
floss$ cvs watch off
```

Una sugerencia para diagnosticar errores con los watches: a veces puede clarificar mucho simplemente el entrar en el repositorio y examinar el fichero CVS/fileattr directamente. Véase ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ para más información sobre esto.

Mis ficheros binarios se han corrompido

Nos hemos acordado de usar el modificador -kb al insertarlos? Si no fue así, CVS puede haber realizado expansiones de macros RCS o conversiones de fin de línea. La solución más simple es marcarlos como binarios,

```
floss$ cvs admin -kb ejemplo.gif
```

y luego enviar una versión corregida del fichero. CVS no corromperá nuevos envíos de este fichero, puesto que ya sabe que es binario.

CVS no hace correctamente las conversiones de fin de línea

Si hemos ejecutado el cliente CVS en una plataforma no Unix, y no tenemos las conversiones correctas de final de línea, se deberá normalmente a que hemos añadido los ficheros accidentalmente como binarios (opción -kb). Esto puede corregirse en el repositorio con el comando:

```
floss$ cvs admin -kkv FICHERO
```

El modificador -kkv solicita hacer la expansión de macros normal y las conversiones de fin de línea (internamente, CVS se confunde con la diferencia entre la expansión de macros y la conversión de fin de línea. Esta confusión da lugar a que las opciones de -k siempre controlan ambos aspectos a la vez).

Por desgracia, este comando de administración solo corrige el fichero en el repositorio, es decir, nuestra copia local seguirá considerándose como binaria. Siempre podemos editar a mano el fichero CVS/Entries eliminando la opción -kb de la línea correspondiente.

Cómo se borra un directorio del proyecto?

Bien, no podemos borrar el directorio realmente, pero podemos borrar todos los ficheros que haya dentro (primero hacemos cvs remove y luego commit). Una vez que el directorio está vacío, se eliminará de las copias de trabajo usando la opción -P durante cualquier actualización.

Puedo copiar ficheros .cvspass o parte de ellos?

Sí, claro. Se pueden copiar de una máquina a otra, o se pueden transferir líneas individuales de un fichero .cvspass a otro. Para servidores con acceso lento, esto puede ser más rápido que hacer cvs login en cada máquina.

Recordemos que si transportamos un fichero .cvspass entre dos máquinas con diferentes convenciones de fin de línea, probablemente no funcionarán (por supuesto, siempre se puede corregir este problema a mano).

Acabo de enviar algunos ficheros con un mensaje histórico incorrecto

Para resolver esto no hay que editar nada del repositorio. Simplemente ejecutemos cvs admin con la opción -m. Recuérdese no dejar espacios entre -m y el argumento, y acitar el mensaje histórico tal como se haría normalmente:

```
floss$ cvs admin -m1.17:'Recientes mejoras en mi mejor programa.' hola.c
```

Necesito mover ficheros sin perder el histórico de revisiones

En el repositorio, copiemos (no movamos) los ficheros RCS al nuevo lugar deseado. Esto debe respetar sus antiguas localizaciones. Ahora, desde una copia de trabajo, hagamos lo siguiente:

```
floss$ rm ficheroviejo1 ficheroviejo2 ...
floss$ cvs remove ficheroviejo1 ficheroviejo2 ...
floss$ cvs commit -m 'Ficheros movidos ...'
```

Cuando la gente haga actualizaciones, CVS borrará los ficheros viejos y los creará en sus nuevos sitios, tal como si se hubiera hecho la operación de añadir normalmente (excepto por el hecho de que se siguen usando los números de revisión anteriores).

Como puedo obtener la lista de todas las etiquetas del proyecto?

Actualmente no hay forma de hacerlo con CVS. Es algo demandado por los usuarios y se espera que en futuras versiones esté disponible. Puede que pronto (incluso ya) existe un comando `cvs tags` o similar.

Hasta entonces, hay aproximaciones. Podemos ejecutar cvs log -h y leer las secciones de la salida con la cabecera `symbolic names:`. O bien, si estamos en la máquina del repositorio, podemos mirar en el principio de algunos ficheros RCS. Todas las etiquetas (rama y no rama) se listan en el campo `symbols`:

```
floss$ head /usr/local/nuevorep/hola.c,v
head 2.0;
access;
symbols
Release_1_0:1.22
Exotic_Greetings-2:1.21
merged-Exotic_Greetings-1:1.21
Exotic_Greetings-1:1.21
merged-Exotic_Greetings:1.21
Exotic_Greetings-branch:1.21.0.2
Root-of-Exotic_Greetings:1.21
start:1.1.1.1
jrandom:1.1.1;
locks; strict;
comment @ * @;
```

Como obtener una lista de todos los proyectos del repositorio?

Al igual que sucede con la lista de etiquetas, no está implementado en la versión actual de CVS, pero se entiende que lo estará pronto. Imagino el comando similar a cvs list, con

una forma abreviada cvs ls, y probablemente ambos analizarán los módulos y listarán los subdirectorios.

Por ahora, examinando el fichero CVSROOT/modules (directamente o ejecutando cvs checkout -c) podemos conseguirlo. Sin embargo, si nadie ha hecho un módulo para un proyecto, no veremos nada sobre ese proyecto en el fichero modules.

Algunos comandos fallan en remoto pero no en local; cómo lo depuramos?

A veces hay un problema de comunicación entre el cliente y el servidor. Y puede ser un error de CVS.

CVS proporciona un mecanismo de trazar el protocolo entre el cliente y el servidor. Antes de ejecutar el comando en la máquina local (con la copia de trabajo), crear la variable `CVS_CLIENT_LOG`. En un shell de Bash se haría así:

```
floss$ CVS_CLIENT_LOG=clog; export CVS_CLIENT_LOG
```

Una vez creada la variable, CVS almacenará las comunicaciones entre cliente y servidor en dos ficheros con el nombre basado en el valor de la variable anterior:

```
floss$ ls
CVS/        LEAME.txt    a-subdir/    b-subdir/    prueba.gif     hola.c
floss$ cvs update
? clog.in
? clog.out
cvs server: Updating .
cvs server: Updating a-subdir
cvs server: Updating a-subdir/subsubdir
cvs server: Updating b-subdir
floss$ ls
CVS/              a-subdir/    clog.in     prueba.gif
LEAME.txt        b-subdir/    clog.out    hola.c
floss$
```

El fichero '`clog.in`' contiene lo enviado por el cliente al servidor, y el fichero '`clog.out`' contiene los mensajes del servidor al cliente. Vemos por ejemplo el contenido de clog.out:

```
Valid-requests Root Valid-responses valid-requests Repository           \
Directory Max-dotdot Static-directory Sticky Checkin-prog Update-prog   \
Entry Kopt Checkin-time Modified Is-modified UseUnchanged Unchanged     \
Notify Questionable Case Argument Argumentx Global_option Gzip-stream   \
wrapper-sendme-rcsOptions Set expand-modules ci co update diff log add  \
remove update-patches gzip-file-contents status rdiff tag rtag import   \
admin export history release watch-on watch-off watch-add watch-remove  \
watchers editors init annotate noop
ok
M ? clog.in
M ? clog.out
E cvs server: Updating .
E cvs server: Updating a-subdir
E cvs server: Updating a-subdir/subsubdir
E cvs server: Updating b-subdir
```

```
ok
```

El fichero clog.in es más complicado, puesto que incluye números de versión y otras informaciones por cada fichero.

No podemos dedicar espacio aquí a documentar el protocolo, pero podemos leer las páginas Info de `cvsclient` que vienen con el paquete de CVS para más información. Comprobaremos que aunque no siempre nos dé una respuesta, mirar el histórico del protocolo puede darnos una buena pista.

Mi problema no está explicado en este capítulo

Lo mejor es enviar una descripción del problema a la lista de discusión sobre CVS, `info-cvs@gnu.org`. Los miembros están dispersos por todo el mundo, y por tanto a todas horas suele haber alguien que nos ayudará casi de inmediato. Para apuntarse a la lista hay que enviar un mensaje a `info-cvs-request@gnu.org`. Se agradecerá que nosotros también ayudemos a resolver problemas a los demás.

Creo que he descubierto un bug en CVS; qué hago?

CVS no es perfecto ... si hemos intentado consultar el manual o preguntar por ahí y aun creemos que es un bug, podemos hacer lo siguiente:

Hay que enviar una descripción lo más completa posible del error a `bug-cvs@gnu.org`. A esta lista también podemos suscribirnos escribiendo a `bug-cvs-request@gnu.org`. Hay que incluir en nuestra consulta los números de versión (cliente y servidor) y el modo de reproducir el error.

Si hubiéramos escrito un parche para corregir el error, incluyámoslo y mencionémoslo en la línea del asunto del mensaje. Los desarrolladores lo agradecerán infinitamente.

(En el manual de Cederqvist, en el nodo *BUGS*, encontraremos más detalles sobre cómo seguir estos procedimientos. También hay información en el fichero HACKING de la distribución del código fuente).

He añadido una característica a CVS; a quién la envío?

Al igual que con los errores, enviaremos el parche a `bug-cvs@gnu.org`. Antes tenemos que asegurarnos de que hemos leído el mencionado fichero HACKING.

Como puedo mantenerme informado de las novedades en CVS?

Las técnicas de resolución de problemas y los errores conocidos descritos en este capítulo son para la versión de CVS 1.10.7 (aproximadamente). Pero el mundo CVS se mueve rápidamente. Mientras que escribía los últimos capítulos, el mantenimiento de CVS pasó de Cyclic Software a SourceGear Inc (`http://www.sourcegear.com`), quienes habían comprado Cyclic. SourceGear ha anunciado públicamente su intención de participar activamente en el equipo de desarrolladores de CVS y ha recibido la aprobación de la gente de Cyclic, quienes eran más o menos los que lo lideraban hasta ahora (la dirección `http://www.cyclic.com` seguirá funcionando, sin embargo, de manera que todas las URL dadas aquí aún valen).

SourceGear está, en este preciso momento, ocupada organizando y puliendo varios parches que estaban circulando por ahí, con la intención de incorporar los que puedan a CVS. Algunos parches probablemente corregirán errores vistos aquí, y otros añadirán nuevas opciones a los usuarios.

La mejor forma de mantenerse informado de la evolución es leer el fichero NEWS de la distribución de CVS, vigilando además las listas de correo y buscando cambios en el manual de Cederqvist y en la versión en línea, en inglés, de este libro (`http://cvsbook.red-bean.com`).

Referencia de CVS

This chapter is a complete reference to CVS commands, repository administrative files, keyword substitution, run control files, working copy files, and environment variables – everything in CVS as of CVS version 1.10.7 (more accurately, as of August 20, 1999).

Ordenes y Opciones

Esta sección es una referencia de todos los mandatos CVS. Si usted no está familiarizado con las convenciones sintácticas compartidas por la mayoría de los mandatos CVS, se sugiere leer las subsecciones relevantes antes de echar un vistazo a cualquier mandato en particular.

Organización y Convenciones

Esta sección está organizada alfabéticamente para hacerle fácil encontrar un mandato u opción particular. Las siguiente convenciones son usadas:

Los argumentos a mandatos y opciones tienen todos sus caracteres en may\'usculas en la sinopsis que encabeza cada explicación. (Nota: en la versión treeware del libro, los meta-argumentos están en cursiva además de estar en mayúsculas; debido a la limitación de las fuentes estándar de terminal, he omitido la cursiva aquí).

Los ítems opcionales aparecen entre corchetes: `[]`. (Esto funciona correctamente porque los corchetes no son usados en la sintaxis de CVS).

Si debe elegir una opción de una lista, las elecciones se separan por barras, como esto: `x|y|z`. (Y, por tanto, las barras (`/`) se deberían interpretar literalmente – no dividen elecciones de una lista).

Los plurales y los puntos suspensivos indican múltiples opciones, usualmente separadas por espacio. Por ejemplo, FILES significa uno o más ficheros, pero [FILES] significa cero o más ficheros. La entrada [&MOD...] significa un ampersand seguido inmediatamente por el nombre de un módulo, depués espacio, después tal vez otro ampersand-módulo, y así sucesivamente, cero o más veces. (Los puntos suspensivos se usan porque un plural podría no dejar claro si el ampersand se necesita sólo la primera vez o todas para cada módulo).

Cuando un plural está entre paréntesis, como en FILE(S), significa que aunque técnicamente puede haber dos o más ficheros, usualmente sólo hay uno.

REV se utiliza habitualmente para referir a un argumento de revisión. Esto es normalmente, bien un número de revisión, bien un nombre de etiqueta. Hay pocos sitios en CVS en que puede usar uno, pero no el otro, y estos sitios están señalados en el texto.

Patrones Generales En Los Mandatos CVS

Los mandatos de CVS poseen este formato:

```
cvs [OPCIONES_GLOBALES] MANDATO [OPCIONES] [FICHEROS]
```

El segundo juego de opciones se llama a veces *opciones de mandato*. Debido a que hay tantos, sin embargo, lo llamaré simplemente "opciones" en la mayoría de los sitios para ahorrar espacio.

Muchos mandatos están pensados para ejecutarse en la copia de trabajo y, por tanto, pueden ser invocados sin argumentos de fichero. Estos mandatos se ejecutarán en todos los ficheros del directorio actual e inferiores. Así, cuando me refiera al "fichero" o "ficheros" en el texto, estoy hablando acerca de los ficheros en los que CVS actúa. Dependiendo de cómo invoca CVS, estos ficheros pueden, o no, haber sido mencionados explícitamente en la línea de mandatos.

Formatos de Fecha

Muchas opciones toman un argumento de fecha. CVS acepta una gran variedad de formatos de fecha – demasiados para ser listados aquí. Cuando dude, use el formato estándar ISO 8601:

```
1999-08-23
```

Esto significa "23 de agosto de 1999" (de hecho, en inglés "23 August 1999" es una especificador de fecha perfectamente válido también, siempre y cuando recuerde encerrarlo entre comillas). Si necesita la hora de un día también, puede hacer esto:

```
"1999-08-23 21:20:30 CDT"
```

Puede incluso usar ciertas construcciones en inglés, como "now" (hoy), "yesterday" (ayer), y "12 days ago" (hace doce días). En general, puede experimentar con seguridad con formatos de fecha; si CVS entiende su formato, muy fácilmente lo entenderá en el modo en que usted trataba de expresarlo. Si no lo entiende, saldrá con un error inmediatamente.

Opciones Globales

Aquí están todas las opciones globales de CVS.

`--allow-root=REPOSITORY`

La primera opción global alfabéticamente es una que no se usa nunca en la línea de mandatos. La opción –alow-root se usa con el mandato `pserver` para permitir acceso acreditado al repositorio dicho (que es el nivel alto de un repositorio, como '`/usr/local/newrepos`', no un subdirectorio de proyecto como '`/usr/local/newrepos/myproj`').

Esta opción global no se usa virtualmente nunca en la línea de mandatos. Normalmente el único sitio donde lo usaría es en los ficheros '`/etc/inetd.conf`' (ver ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩), que es también casi el único sitio donde el mandato `pserver` se usa.

Cada repositorio a ser accedido vía `cvs pserver` en un servidor dado, necesita la correspondiente opción –allow-root en '`/etc/inetd.conf`'. Éste es un dispositivo de seguridad, pensado para asegurar que cualquiera no puede usar un pserver de CVS para conseguir acceso a repositorios privados.

(Ver ⟨undefined⟩ [El servidor de autentificacion de contrasen˜as], page ⟨undefined⟩ también en el nodo *Servidor Acreditado por Clave* en el manual Cederqvist.)

`-a`

Esto acredita todas las comunicaciones con el servidor. Esta opción no tiene efecto a no ser que se esté conectado vía servidor GSSAPI (gserver). Las conexiones GSSAPI no se cubren en este libro, porque son todavía usadas raramente (aunque esto puede cambiar). (Ver las notas *Opciones Globales* y *Acreditación GSSAPI* en el manual Cederqvist para obtener más información.)

`-b` (Caído en desuso)

Esta opción especifica formalmente el directorio donde los ejecutables del RCS se encuentran. CVS ahora implementa las funciones RCS internamente, por lo que esta opción no surte efecto (se mantiene sólo por compatibilidad retrospectiva).

`-d` REPOSITORIO

Esto especifica el repositorio, que puede ser una ruta absoluta o una expresión más compleja involucrando método de conexión, nombre de usuario, servidor y ruta. Si es una expresión especificando un método de conexión, la sintaxis general es:

```
:MÉTODO:USUARIO@NOMBRE_DEL_SERVIDOR:RUTA_AL_REPOSITORIO
```

Aquí hay ejemplos usando cada uno de los métodos de conexión:

`:ext:jcualquiera@floss.red-bean.com:/usr/local/newrepos` – Conecta usando `rsh`, `ssh`, o algún otro programa de conexión externo. Si la variable de entorno `$CVS_RSH` no está especificada, la opción por omisión es `rsh`; de otro modo, usa el valor de esta variable.

`:server:jcualquiera@floss.red-bean.com:/usr/local/newrepos` – Como `:ext:`, pero usa la implementación interna de rsh. (Esto puede no estar disponible en todas las plataformas.)

`:pserver:jcualquiera@floss.red-bean.com:/usr/local/newrepos` – Conecta usando el servidor de acreditación de claves (ver ⟨undefined⟩ [El servidor de autentificacion de contrasen˜as], page ⟨undefined⟩ en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩; ver también el mandato ⟨undefined⟩ [login], page ⟨undefined⟩.)

`:kserver:jrandom@floss.red-bean.com:/usr/local/newrepos` – Conecta usando acreditación Kerberos.

`:gserver:jrandom@floss.red-bean.com:/usr/local/newrepos` – Conecta usando acreditación GSSAPI.

`:fork:jcualquiera@floss.red-bean.com:/usr/local/newrepos` – Conecta a un repositorio local, pero usando el protocolo de red cliente/servidor en vez de acceder directamente a los ficheros del repositorio. Esto es útil para comprobar y depurar comportamientos de CVS en remoto desde su máquina local.

`:local:jcualquiera@floss.red-bean.com:/usr/local/newrepos` – Accede a un repositorio local directamente, como si se diera sólo la ruta absoluta al respositorio.

-e EDITOR

Invoca EDITOR para su mensaje de entrega, si el mensaje de entrega no se especifica en la línea de mandatos con la opción -m. Normalmente, si no da un mensaje con la opción -m, CVS invoca el editor basado en las variables de entorno `$CVSEDITOR`, `$VISUAL`, o `$EDITOR`, que comprueba en este orden. Fallado esto, invoca el editor popular de Unix `vi`.

Si pasa tanto la opción -e como la -m en una entrega, el -e no se tiene en cuenta a favor del mensaje de entrega dado en la línea de mandatos (de este modo es seguro usar -e en un fichero '`.cvsrc`').

-f

Esta opción global suprime la lectura del fichero '`.cvsrc`'.

--help [MANDATO] o -H [MANDATO]

Estas dos opciones son sinónimas. Si no se especifica MANDATO, se imprime un mensaje de uso básico a la salida estándar. Si se especifica MANDATO, se imprime un mensaje de uso para ese mandato.

--help-options

Imprime una lista con todas las opciones globales de CVS, con breves explicaciones.

--help-synonyms

Imprime una lista de mandatos CVS y sus formatos cortos ("up" para "update", y así sucesivamente).

-l

Suprime el registro del mandato en el fichero '`CVSROOT/history`' en el repositorio. El mandato se ejecuta normalmente, pero no se realiza ninguna archivación en el fichero de historial.

-n

No cambia ningún fichero en la copia de trabajo o en el repositorio. En otras palabras, el mandato se ejecuta como una "ejecución en seco" – CVS corre a través de la mayoría de los pasos del mandato pero evita cualquier clase de ejecución.

Ésto es útil cuando quiere ver qué habría hecho el mandato si usted lo hubiera ejecutado. Un escenario común es cuando quiere ver qué ficheros de su directorio de trabajo han sido alterados, pero no una actualización completa (que podrí traer cambios del repositorio). Ejecutando `cvs -n update`, puede ver un sumario de qué ha sido hecho localmente, sin cambiar su copia de trabajo.

-q

Esto pide a CVS ser moderadamente silencioso, suprimiendo la impresión de mensajes de información no importantes. Qué es considerado "importante" depende del mandato. Por ejemplo, en actualizaciones, los mensajes que CVS imprime normalmente al entrar en cada subdirectorio de la copia de trabajo se suprimen, pero los mensajes de estado de una línea para los ficheros modificados o actualizados se siguen imprimiendo.

-Q

Esto pide a CVS ser muy silencioso suprimiendo toda la salida excepto los que son absolutamente necesario para completar el mandato. Los mandatos cuyo único propósito es producir alguna salida (como `diff` o `annotate`), de hecho, siguen dando esa salida. Sin embargo, los mandatos que podrían tener un efecto independiente de cualquier mensaje que pudieran imprimir (como `update` o `commit`) no imprimen nada.

-r

Hace que los ficheros de trabajo sean creados como de sólo lectura (el mismo efecto que configurando la variable de entorno `CVSREAD`).

Si usted pasa esta opción, las obtenciones y las entregas hacen los ficheros de sólo lectura en su copia de trabajo (asumiendo que su sistema operativo lo permita). Francamente, no sé por qué alguien podría querer usar alguna vez esta opción.

-s VARIABLE=VALOR

Esto asigna el VALOR a la variable interna de CVS llamada VARIABLE.

En el lado del repositorio, los fichero disparadores '`CVSROOT/*info`' puede expandir tales variables a valores que fueron asignados en la opción -s. Por ejemplo, si '`CVSROOT/loginfo`' contiene una línea como esta

```
miproyecto  /usr/local/bin/foo.pl ${=PEZ}
```

y alguien ejecuta una entrega desde una copia de trabajo miproyecto así

```
floss$ cvs -s PEZ=carpa commit -m "arreglado el bug cebo"
```

el script '`foo.pl`' se invoca con `carpa` como un argumento. Note la sintaxis chula, así: El signo del dólar, igual y las llaves son todas necesarias – si alguno de ellos falta, la expansión no toma lugar (al menos no como se trataba). Los nombres de variables sólo pueden contener alfanuméricos y subrayados. Aunque no se requiere que estén en mayúsculas, la mayoría de la gente parece seguir esta convención.

Puede usar el indicador -s tantas veces como quiera en un mandato simple. Sin embargo, si el script disparador se refiere a variables que no son asignadas en una invocación particular de CVS, el mandato también tiene éxito, pero ninguna de las variables se expande, y el usuario ve un aviso. Por ejemplo, si loginfo tiene esto

```
miproyecto  /usr/local/bin/foo.pl  ${=PEZ}  ${=AVE}
```

pero el mismo mandato de antes se ejecuta

```
floss$ cvs -s PEZ=carpa commit -m "arreglado el bug cebo"
```

la persona que ejecuta el mandato ve un mensaje de aviso como este (puesto al final de la salida)

```
loginfo:31: no such user variable ${=AVE}
```

(loginfo:31: variable de usuario inexistente) y el script '`foo.pl`' se invoca sin argumentos. Pero si se ejecuta este mandato

```
floss$ cvs -s PEZ=carpa -s AVE=buitre commit -m "arreglado el bug cebo"
```

aquí no habría aviso, y tanto `${=PEZ}` como `${=AVE}` en loginfo estarían correctamente expandidos. En cualquier caso, la entrega en sí misma, tendría éxito.

Aunque estos ejemplos usan todos `commit`, la expansión de variables puede hacerse con cualquier mandato CVS que pueda ser notificado en un fichero disparador '`CVSROOT/`' – que es por lo que la opción -s es global.

(Vea la sección ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ más adelante en este capítulo para obtener más detalles sobre la expansión de variables en ficheros disparadores.)

-T DIRECTORIO

Guarda cualquier fichero temporal en el DIRECTORIO en vez de donde CVS lo pone normalmente (específicamente, esto sustituye el valor de la variable de entorno `$TMPDIR`, si existiera). DIRECTORIO debería ser una ruta absoluta.

Esta opción es útil cuando usted no tiene que permiso de escritura (y, por tanto, CVS tampoco) a los directorios temporales usuales.

-t

Traza la ejecución del mandato de CVS. Esto hace a CVS imprimir mensajes mostrando los pasos que se dan a lo largo de la ejecución de un mandato. Puede encontrarlo particularmente útil en conjunción con la opción global -n, para prever los efectos de un mandato poco familiar antes de ejecutarlo de verdad. También puede ser útil cuando intenta descubrir por qué un mandato falló.

-v o --version

Hace que CVS imprima información de sus versiones y derechos de copia y después salga sin error.

-w

Hace que los ficheros de trabajo sean creados como lectura-escritura (remplaza cualquier configuración de las variables de entorno `$CVSREAD`). Debido, de todos modos, a que los ficheros se crean por omisión en modo lectura-escritura, esta opción se usa raramente.

Si tanto -r como -w se pasan, -w domina.

-x

Encripta todas las comunicaciones con el servidor. Esta opción no tiene efecto a no ser que esté conectando vía servidor GSSAPI (gserver). Las conexiones GSSAPI no se cubren en este libro, porque son todavía raramente usadas (aunque esto puede cambiar). (Ver los nodos *Opciones Globales* y *Acreditación GSSAPI* en el manual Cederqvist para ampliar información.)

-z NIVEL-GZIP

Ajusta el nivel de compresión en las comunicaciones con el servidor. El argumento NIVEL-GZIP debe ser un número entre el uno y el nueve. El nivel uno es compresión mínima (muy rápida, pero poca compresión). El nivel nueve es la compresión más alta (aunque es mucho tiempo de CPU, pero asegura que comprime los datos). El nivel nueve es solamente útil en conexiones de red muy rapidas. La mayor parte de la gente encuentra los niveles entre el tres y el cinco los más benificiosos.

Un espacio entre -z y su argumento es opcional.

add

Sinopsis: add [OPCIONES] FICHEROS

Nombres alternativos – ad, new

Requiere – Copia de trabajo, repositorio

Cambia – Copia de trabajo

Añade un fichero o ficheros nuevos a un proyecto existente. Aunque se conecta con el repositorio para obtener confirmación, el fichero no aparece en el acto en él hasta que se realiza la siguiente llamada a commit (Ver también ⟨undefined⟩ [remove], page ⟨undefined⟩ e ⟨undefined⟩ [import], page ⟨undefined⟩.)

Opciones:

-kCLAVE_DE_MODO_DE_SUSTITUCIÓN – Especifica que el fichero va a ser almacenado con la clave de sustitución de RCS dada. No hay espacio entre -k y su argumento. (Ver la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en este capítulo para tener una lista completa de modos válidos y ejemplos.)

-m MENSAJE – Graba el MENSAJE como mensaje de creación, o descripción, para el fichero. Éste es diferente de un mensaje de registro por revisión – cada campo tiene una sola descripción. Las descripciones son opcionales.

Hasta la versión 1.10.7, hay un bug en CVS por el que la descripción se pierde se añade un fichero via CVS cliente/servidor. El resto del proceso de añadido funciona correctamente, sin embargo, si sirve de consuelo.

admin

Synopsis: admin [OPCIONES] [FICHEROS]

Nombres alternativos – adm, rcs

Requiere – Copia de trabajo, repositorio

Cambios – Repositorio

Este mandato es una interfaz a las distintas tareas de administración – especialmente, tareas aplicables a ficheros RCS individuales en el repositorio, como cambiar la clave de sustitución o cambiar un mensaje de registro después de que haya sido entregado.

Aunque admin se comporta recursivamente, si no hay ficheros dados como argumentos, normalmente usted querrá nombrarlos explícitamente. Es muy raro que un mandato admin suelto tenga sentido cuando se aplica a todos los ficheros en un proyecto, o incluso un directorio. De este modo, cuando las siguientes explicaciones se refieran al "fichero", quere decir el fichero o (raramente) los ficheros pasados como argumentos al mandato admin.

Si hay un grupo de sistema llamado `cvsadmin` en la máquina del repositorio, sólo los miembros de este grupo pueden ejecutar admin (con la excepción de la orden `cvs admin -k`, que siempre se permite). Así, usted puede impedir admin para todos los usuarios configurando el grupo para no tener usuarios.

Opciones:

-AFICHERO_VIEJO – (En desuso) Adjunta la lista de accesos RCS del FICHERO_VIEJO a la lista de accesos del fichero que es el argumento de admin. CVS no tiene en cuenta las listas de acceso RCS, así que esta opción es inútil.

-a USUARIO1 [,USUARIO2...] – (En desuso) Adjunta los usuarios en la lista separada por comas a la lista de acceso del fichero. Como -A, esta opción es inútil en CVS.

-bREV – Especifica la revisión de la rama por omisión del fichero (usualmente el tronco) a REV. No necesita normalmente esta opción, pero podría usarla para volver a una versión de un vendedor si está usando ramas de vendedores. No debe haber espacio entre la opción -b y su argumento.

-cPREFIJO_DE_COMENTARIO – (En desuso) Especifica la cabecera de comentario del fichero a PREFIJO_DE_COMENTARIO. El encabezamiento del comentario no se usa por CVS ni incluso por recientes versiones de RCS; así, esta opción es inútil y se incluye sólo por compatibilidad retrospectiva.

-eUSUARIO1[,USUARIO2...] – (En desuso) Quita los nombres de usuario que aparecen en la lista separada por comas de la lista de acceso del fichero RCS. Como -a y -A, esta opción es ya inútil en CVS.

-i o -I – Estas dos han caído tan en desuso que ni siquiera voy a contarle qué hacían. (Ver el manual Cederqvist si siente curiosidad).

-kMODO – Especifica la clave de sustitución por omisión del fichero a MODO. Esta opción se comporta como la opción -k para añadir, sólo le proporciona una manera de cambiar el modo de un fichero después de que haya sido añadido. (Ver la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en el capítulo para ver modos válidos). No debe haber espacio entre -k y su argumento.

-L – Configura el candado a `strict`. (Ver -l abajo.)

-l[REV] – Cierra la revisión del fichero a REV. Si REV es omitido, cierra la última revisión en la rama por omisión (usualmente el tronco). Si REV es una rama, cierra la última revisión de la rama.

La intención de esta opción es proporcionarle un modo de hacer *obtenciones reservadas*, donde sólo un usuario a la vez puede estar editando un fichero. No estoy seguro de lo útil que es esto realmente, pero si quiere probarlo, debe probablemente hacerlo en

conjunción con el script '`rcslock.pl`' en el directorio de la distribución '`contrib/`'. Ver comentarios en este fichero para obtener más información. Entre otras cosas, estos comentarios indican que el bloqueo debe ser configurado a `strict`. (Ver -L.) No hay espacio entre -l y su argumento.

-mREV:MENSAJE – Cambia el registro de mensajes para la revisión REV a MENSAJE. Muy útil – junto con -k, esta es probablemente la opción de administración más usada. No hay espacio entre la opción y los argumentos o alrededor de las comas entre dos argumentos. De hecho, MENSAJE puede contener espacios dentro (en tal caso, recuerde rodearlo por comillas para que el shell sepa que todo ello es una sola cosa).

-NNOMBRE[:[REV]] – Igual que -n, excepto que fuerza el remplazo de cualquier asignamiento existente del nombre simbólico NOMBRE, en vez de salir con un error.

-nNOMBRE[:[REV]] – Éste es un interfaz genérico a asignar, renombrar y borrar etiquetas. No hay razón, que yo sepa, de preferirlo al mandato tag y las variadas opciones disponibles ahí (-d, -r, -b, -f, y esas). Recomiendo usar, en cambio, el mandato tag. El NOMBRE y la REVisión opcional pueden ser combinadas de las siguientes formas:

> Si sólo se da el argumento NOMBRE, el nombre simbólico (etiqueta) llamado NOMBRE se borra.
>
> Si NOMBRE: se da pero no REV, NOMBRE se asigna a la última revisión en la rama por omisión (usualmente el tronco).
>
> Si NOMBRE:REV se da, el NOMBRE se asigna a esa revisión. REV puede ser un nombre simbólico en sí mismo, en este caso se traduce a un número de revisión primero (puede ser un número de rama).
>
> Si REV es un número de rama y le sigue un punto (`.`), NOMBRE se adjunta a la revisión más alta de esa rama. Si REV es simplemente $, NOMBRE se adjunta a los números de revisión encontrados en las cadenas de claves en los ficheros de trabajo.
>
> En todos los casos donde NOMBRE se asigna, CVS sale con un error si había una etiqueta llamada NOMBRE en el fichero (pero vea -N). No hay espacios entre -n y sus argumentos.

-oRANGO – Borra las revisiones especificadas por RANGO (también conocido como "anticuar", de ahí la -o). El rango puede ser especificado de uno de las siguientes modos:

> REV1::REV2 – Colapsa todas las revisiones intermedias entre REV1 y REV2, de modo que el historial de revisiones vaya directamente de REV1 a REV2. Después de esto, cualquier revisión entre las dos ya no existirá, y habrá un salto discontinuo en la secuencia de números de revisiones.
>
> ::REV – Colapsa todas las revisiones entre el principio de la rama de REVisión (que puede ser el principio del tronco) y la REVisión, no inclusive, por supuesto. REV es la primera revisión de esta línea.
>
> REV:: – Colapsa todas las revisiones entre REV y el final de su rama (que puede ser el tronco). REV es entonces la última revisión de esta línea.
>
> REV – Borra la revisión REV (-o1.8 sería equivalente a -o1.7::1.9).

REV1:REV2 – Borra las revisiones de REV1 a REV2, incluídas. Deben estar en la misma rama. Después de esto, no puede recuperar REV1, REV2, o cualquier otra de las revisiones entre ellas.

:REV – Borra las revisiones entre el comienzo de la rama (o tronco) a REV, incluída. (Ver el aviso precedente.)

Note que las revisiones que se borran pueden tener ramas o bloqueos. Si cualquiera de las revisiones tiene nombres simbólicos adjuntos, deberá borrarlos primero con tag -d o admin -n. (De hecho, ahora mismo CVS sólo protege contra el borrado de revisiones con nombres simbólicos si está usando una de las sintaxis ::, pero las sintaxis con coma tal vez cambien pronto a este comportamiento también).

En vez de usar esta opción para deshacer una entrega mala, puede entregar una nueva revisión de deshaga el cambio erróneo. No hay espacios entre -o y sus argumentos.

-q – Dice a CVS que se ejecute silenciosamente – no se imprimen mensajes de diagnóstico (simplemente como la opción global -q).

-sESTADO[:REV] – Configura el atributo de estado de la revisión REV a ESTADO. Si REV se omite, la última revisión en la rama por omisión (usualmente el tronco) se usa. Si REV es una etiqueta de rama o número, se usa la última revisión de esta rama.

Cualquier cadena de letras o números es aceptable para ESTADO; unos estados comúnmente usados son Exp para Experimental, Stab para Estable y Rel para Lanzamiento. (De hecho, CVS ajusta el estado a Exp cuando un fichero se crea). Note que CVS usa el estado "dead" para sus propios propósitos, así que no lo especifique.

Los estados se muestran en la salida de registro de CVS y en las claves RCS $Log y $State en los ficheros. No hay espacio entre -s y sus argumentos.

-t[FICHERO_DESC] – Remplaza la descripción (mensaje de creación) para el fichero con contenidos FICHERO_DESC, o lo lee de la entrada estándar si no se especifica FICHERO_DESC.

Esta útil opción, desafortunadamente, no funciona actualmente en CVS cliente/servidor. Además, si lo itenta en cliente/servidor y omite FICHERO_DESC, cualquier descripción existente para el fichero es cortada y remplazada por una cadena vacía. Si necesita reescribir la descripción de un fichero, hágalo bien usando sólo CVS local en la misma máquina que el repositorio o -t-CADENA (ver adelante). No hay espacio entre -t y su argumento. FICHERO_DESC puede no comanzar con guión (-).

-t-CADENA – Como -t, excepto que CADENA es tomada directamente como la nueva descripción. CADENA puede contener espacios, en cuyo caso debe rodearla por comillas. A diferencia de la otra sintaxis para -t, esta funciona tanto en cliente/servidor como localmente.

-U – Configura el bloqueo a no-estricto. (Ver opciones -l y -L, discutidas anteriormente.)

-u[REV] – Libera la revisión REV. (Ver -l). Si se omite REV, CVS libera el último bloqueo mantenido por el solicitante. Si algún otro que el dueño de un bloqueo lo rompe, un mensaje por correo electrónico se envía al dueño original del bloqueo. El contenido de este mensaje se solicita en la entrada estándar de la persona que rompe el bloqueo. No hay espacio entre -u y su argumento.

-VNÚMERO_DE_VERSIÓN_CVS – (En desuso) Esto solía ser un modo de pedir a CVS producir ficheros RCS aceptables para versiones anteriores de RCS. Ahora el formato

RCS usado por CVS está quedándose lejos del formato RCS usado por RCS, así que la opción es inútil. Especificarla acaba en un error.

-xSUFIJO – (En desuso) Teóricamente, esto le da un modo de especificar el sufijo de los nombres de fichero RCS. Sin embargo, CVS y las herramientas relacionadas dependen todos del sufijo por omisión (,v), así que esta opción no hace nada.

annotate

Sinopsis: annotate [OPCIONES] [FICHERO]

Nombre alternativo – ann

Requiere – Copia de trabajo, repositorio

Cambia – Nada

Muestra información de quién ha sido el último en modificar cada línea de salida correspondiente a una línea del fichero. De izquierda a derecha, la línea muestra el número de revisión de la última modificación de esa l\'inea, una expresión entre paréntesis conteniendo el usuario y la fecha de la modificación, una coma, y los contenidos de la línea en el fichero.

Por ejemplo, si un fichero tiene este aspecto

```
este es un fichero de prueba
tiene muchas líneas
quiero decir "dos"
```

las anotaciones para este fichero podrían parecer esto

```
1.1          (jcualquiera  22-Aug-99): este es un fichero de prueba
1.1          (jcualquiera  22-Aug-99): tiene muchas líneas
1.2          (jcualquiera  22-Aug-99): quiero decir "dos"
```

de donde puede saber que las primeras dos líneas se escribieron en la revisión inicial, y la última fue añadida o modificada (también por jcualquiera) en la Revisión 1.2.

Opciones:

-D FECHA – Muestra las anotaciones de la última revisión no más antigua que FECHA.

-f – Fuerza el uso de una revisión de la etiqueta especificada o si la fecha no se encuentra. Puede usar esto en combinación con -D o -r para asegurar que hay alguna salida del mandato annotate, incluso si sólo se muestra la revisión 1.1 del fichero.

-l – Local. Ejecutar sólo en el directorio de trabajo actual. No desciende dentro de los subdirectorios.

-R – Recursivo. Desciende dentro de los subdirectorios (por omisión). El punto en la opción -R se hace para sustituir cualquier opción -l puesta en un fichero .cvsrc.

-r REV – Muestra las anotaciones de la revisión REV (puede ser un número de revisión o una etiqueta).

checkout

Sinopsis: checkout [OPCIONES] PROYECTO(S)

Nombres alternativos – co, get

Requiere – Repository

Cambios – Current directory

Obtiene un módulo del repositorio en una copia de trabajo. La copia de trabajo se crea si no existe ya y se actualiza si existe. (Ver también ⟨undefined⟩ [update], page ⟨undefined⟩.)

Opciones:

-A – Borra cualquier etiqueta adhesiva, fechas adhesivas, o -k adhesivas (clave de modo de sustitución RCS). Esto es como la opción -A para actualizar y es probablemente usada más a menudo que con la obtención.

-c – No obtiene; simplemente imprime el fichero '`CVSROOT/modules`', ordenado, en la salida estándar. Es un buen modo de obtener un resumen de qué proyectos están en el repositorio. Sin embargo, un proyecto sin una entrada en modules no aparece (esta situación es muy normal porque el nombre del directorio de alto nivel del proyecto en el repositorio funciona como el nombre por omisión del módulo del proyecto).

-D FECHA – Obtiene la última revisión no más vieja que FECHA. Esta opción es adhesiva, así que no podrá entregar desde la copia de trabajo sin borrar la fecha adhesiva. (Ver -A). Esta opci\on también implica -P, descrita más tarde.

-d DIR – Crea la copia de trabajo en un directorio llamado DIR, en vez de crear el directorio con el mismo nombre que el módulo obtenido. Si obtiene sólo una porción de un proyecto y la porción está ubicada en cualquier parte que no sea el nivel más alto del proyecto, los directorios son omitidos. Puede usar -N para suprimir este comportamiento de colapso de directorios.

-f – Fuerza la obtención de la revisión de cabecera si la etiqueta especificada o la fecha no se encuentra. Se usa muy a menudo en combinación con -D o -R para asegurar que algo siempre se obtiene.

-j REV[:FECHA] o -j REV1[:FECHA] -j REV2[:FECHA] – Une (mezcla) dos líneas de desarrollo. Esto es simplemente como la opción -j para actualizar, donde es más comúnmente usado. (Vea ⟨undefined⟩ [update], page ⟨undefined⟩ para detalles.)

-k MODO – Sustituye la clave RCS de acuerdo con MODO (que puede sustituir los modos por omisión de los ficheros). (Vea la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en este capítulo con los modos válidos.) El modo elegideo será adhesivo – futuras actualizaciones de la copia de trabajo mantendrán este modo.

-l – Local. Obtiene sólo el directorio de alto nivel del proyecto. No procesa subdirectorios.

-N – Suprime el colapso de directorios vacíos con la opción -d (Vea -d.)

-n – No ejecuta ningún programa que fue especificado con -o en '`CVSROOT/modules`'. (Ver la sección ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ más adelante en este cap\'itulo para obtener más información.)

-P – Poda directorios vacíos de la copia de trabajo (como la opción -P de update).

-p – Obtiene a la salida estándar, no a ficheros (como la opción -p de update).

-R – Obtiene también subdirectorios (por omisión). (Ver también la opción -f.)

-r ETIQUETA – Obtiene el proyecto con revisión ETIQUETA (no tendría sentido especificar una revisión numérica para ETIQUETA, aunque CVS se lo permita). Esta opción es adhesiva e implica -P.

-s – Como -c, pero muestra el estado de cada módulo y ordena por estado. (Ver ⟨undefined⟩ [modules], page ⟨undefined⟩ en la sección ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ para ampliar información.)

commit

Synopsis: commit [OPCIONES] [FICHEROS]

Nombres alternativos – ci, com

Requiere – Copia de trabajo, repositorio

Cambia – Repositorio (Y área administrativa de la copia de trabajo)

Entrega los cambios de una copia de trabajo al repositorio.

Opciones:

-F FICHERO_MSJ – Usa los contenidos de FICHERO_MSJ para los mensajes de registro en vez de invocar un editor. Esta opción no puede combinarse con -m.

-f – Fuerza la entrega de una nueva revisión incluso si no se han hecho cambios a los ficheros. `commit` no es recursivo con esta opción (implica -l). Puede forzar la recursión con -R.

El significado de -f no concuerda con su significado usual ("forzar a la revisión de cabecera") en los mandatos de CVS.

-l – Local. Entrega los cambios del directorio actual sólo. No desciende a los subdirectorios.

-m MENSAJE – Usa MENSAJE como mensaje de registro en vez de invocar un editor. No puede ser usado con -F.

-n – No ejecuta ningún programa de módulo. (Ver la sección ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ en este capítulo para obtener más información sobre los programas de módulo.

-R – Entrega los cambios de los subdirectorios al igual que desde el directorio actual (por omisión). Esta opción se usa sólo para contrarrestar el efecto de un -l en '`.cvsrc`'.

-r REV – Entrega hasta la revisión REV, que debe ser, o una rama, o una revisión en el tronco que es más alta que cualquier revisión existente. Al entregar en una rama siempre se va al extremo de la rama (extendiéndola); no puede entregar a una expecífica revisión de una rama. El uso de esta opción configura la nueva revisión como un sticky tag del fichero. Esto puede ser borrado con update -A.

La opción -r REV implica -f también. Una nueva revisión se entrega incluso si no hay cambios que entregar.

diff

Synopsis: diff [OPCIONES] [FICHEROS]

Nombres alternativos – di, dif

Requiere – Copia de trabajo, repositorio

Cambia – Nada

Muestra las diferencias entre dos revisiones (un formato diff de Unix). Cuando es invocado sin opciones, CVS diferencia la revisión base del repositorio entre los (posiblemente no entregados) contenidos de la copia de trabajo. Las revisiones *base* son las últimas revisiones en esta copia recogida del repositorio; note que podría haber incluso revisiones posteriores en el repositorio, si otro entregó cambios pero la copia de trabajo todavía no se ha actualizado. (Ver también ⟨undefined⟩ [rdiff], page ⟨undefined⟩).

Opciones:

-D FECHA – Diferencia entre las últimas revisiones no posteriores a FECHA. Se comporta como -r REV, excepto que usa las fechas en vez de revisiones. (Ver -r para obtener más detalles).

-k MODO – Expande las claves RCS en las diferencias de acuerdo al MODO. (Ver la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ en este capítulo para posibles modos.)

-l – Local. Si no hay ficheros especificados como argumentos, esta opción diferencia en el directorio actual, pero no desciende dentro de los subdirectorios.

-R – Recursivo. Esta opción es la contraria a -l. Este es el comportamiento por omisión, así que la única razón para especificar -R es contrarrestar un -l en un fichero '`.cvsrc`'.

-r REV or -r REV1 -r REV2 – Diferencia entre las revisiones especificadas. Con una opción -r, se diferencian revisiones REV contra su copia de trabajo de este fichero (así cuando múltiples ficheros están siendo diferenciados, REV es casi siempre una etiqueta). Con dos opciones -r, diferencia REV1 entre REV2 para cada fichero (y la copia de trabajo es, por tanto, irrelevante). Las dos revisiones pueden estar en cualquier orden – REV1 no tiene tiene por qué ser una revisión anterior a REV2. La salida refleja las direcciones del cambio. Con ninguna opción -r, muestra las diferencias entre el fichero de trabajo y la revisión en la que está basado.

Opciones de Compatibilidad de Diff

Además de las opciones precedentes, cvs diff también comparte multitud de opciones con la versión GNU del programa de l\'inea de mantados diff estándar. Lo que sigue es una lista completa de estas opciones, junto con una explicación de algunos de los más comúnmente usados. (Ver la documentación del GNU diff para las otras).

```
-0 -1 -2 -3 -4 -5 -6 -7 -8 -9
    --binary
    --brief
    --changed-group-format=ARG
    -c
      -C NL\'INEAS
      --context[=L\'INEAS]
    -e --ed
    -t --expand-tabs
    -f --forward-ed
    --horizon-lines=ARG
    --ifdef=ARG
    -w --ignore-all-space
    -B --ignore-blank-lines
    -i --ignore-case
```

```
-I REGEXP
   --ignore-matching-lines=REGEXP
-h
-b --ignore-space-change
-T --initial-tab
-L ETIQUETA
  --label=ETIQUETA
--left-column
-d --minimal
-N --new-file
--new-line-format=ARG
--old-line-format=ARG
--paginate
-n --rcs
-s --report-identical-files
-p
--show-c-function
-y --side-by-side
-F REGEXP
--show-function-line=REGEXP
-H --speed-large-files
--suppress-common-lines
-a --text
--unchanged-group-format=ARG
-u
  -U NL\'INEAS
  --unified[=L\'INEAS]
-V ARG
-W COLUMNAS
  --width=COLUMNAS
```

Lo que sigue son las opciones del GNU diff más frecuentemente usadas con cvs diff.

-B – No tiene en cuenta las diferencias que son meramente inserción o borrado de líneas vacías (líneas que no contienen nada más que caracteres de espacio).

-b – No tiene en cuenta las diferencias en la cantidad de espacios. Esta opción trata todas las secuencias de espacios como si fueran iguales y no hace caso de los espacios en el final de la línea. Más técnicamente, esta opción colapsa cada secuencia de espacios en la salida a un simple espacio y quita cualquier espacio del final de cada línea, antes de realizar la diferenciación. (Ver también -w).

-c – Muestra la salida en el contexto del formato diff, tomando por omisión tres líneas de contexto por diferencia (en beneficio del programa patch, que requiere como mínimo dos líneas de contexto).

-C NUM – context=NUM – Como -c, pero con NUM líneas de contexto.

-i – Compara sin tener en cuenta mayúsculas/minúsculas. Trata las versiones de mayúsculas y minúsculas de una letra como la misma.

-u – Muestra la salida en un formato unificado diff.

-w – No tiene en cuenta todas las diferencias de espacios, incluso cuando una cara de

la entrada tiene espacios donde las otras no tenían. Esencialmente una versión más fuerte de -b.

edit

Synopsis: edit [OPCIONES] [FICHEROS]

Nombres alternativos – ninguno

Requiere – Copia de trabajo, repositorio

Cambia – Permisos en la copia de trabajo, lista de observaciones en el repositorio

Señala que esta a punto de empezar a editar un fichero o ficheros observados. Atambién le añade como un observador temporal a la lista de observadores del fichero (será quitado cuando haga cvs unedit). (Ver también ⟨undefined⟩ [watch], page ⟨undefined⟩, ⟨undefined⟩ [watchers], page ⟨undefined⟩, ⟨undefined⟩ [unedit], page ⟨undefined⟩ y ⟨undefined⟩ [editors], page ⟨undefined⟩.)

Opciones:

-a ACCIONES – Especifica para qué acciones quiere ser un observador temporal. Las ACCIONES pueden ser edit, unedit, commit, all o none. (Si no usa -a, la observación temporal será para todas las acciones.)

-l – Se\˜ala la edición de los ficheros sólo para el directorio de trabajo actual.

-R – Recursivo (ésto es por omisión). Contrario de b; necesitaría pasar -R sólo para contrarrestar un -l en un fichero '`.cvsrc`'.

editors

Synopsis: editors [OPCIONES] [FICHEROS]

Nombres alternativos – ninguno

Requiere – Copia de trabajo, repositorio

Cambia – Nada

Muestra quién está actualmente editando un fichero observado (Ver también ⟨undefined⟩ [watch], page ⟨undefined⟩, ⟨undefined⟩ [watchers], page ⟨undefined⟩, ⟨undefined⟩ [edit], page ⟨undefined⟩ y ⟨undefined⟩ [unedit], page ⟨undefined⟩.)

Opciones:

-l – Local. Mustra los editores de ficheros en el directorio de trabajo actual sólo.

-R – Recursivo. Muestra los editores para los ficheros en este directorio y sus subdirectorios (por omisión). Necesitará pasar -R para contrarrestar un -l en un fichero '`.cvsrc`', no obstante.

export

Synopsis: export [OPCIONES] PROYECTO(S)

Nombres alternativos – exp, ex

Requiere – Repositorio

Cambia – Directorio actual

Exporta los ficheros del repositorio para crear un árbol de proyecto que no es una copia de trabajo (no tiene subdirectorios CVS/ administrativos). Útil principalmente para empaquetar distribuciones.

Opciones:

-D FECHA – Exporta las últimas revisiones no posteriores a FECHA.

-d DIR – Exporta en DIR (en otro caso, por omisión es en el nombre del módulo).

-f – Fuerza el uso de revisiones de cabecera, si una etiqueta dada o una fecha resulta en que no se encuentra nada (para usar con -D o -r).

-k MODO – Expande una clave RCS de acuerdo al MODO. (Ver la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en este caítulo.)

-l – Local. Exporta sólo el nivel alto del proyecto, no los subdirectorios.

-N – No "colapsa" directorios itermedios vacíos. Esta opción es como la opción -N para la obtención (see ⟨undefined⟩ [checkout], page ⟨undefined⟩).

-n – No ejecuta un programa de módulo como podría ser especificado en '`CVSROOT/modules`'. (Ver ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ más adelante en este campítulo para obtener más detalles acerca de esto).

-P – Borra directorios vacíos (como la opción -P para la obtención o actualización).

-R – Recursivo. Exporta todos los subdirectorios del proyecto (por omisión). La única razón para especificar -R es para contrarrestar un -l en un fichero '`.cvsrc`'.

-r REV – Exporta la revisión REV. Rev es casi ciertamente un nombre de etiqueta, no una revisión numérica.

gserver

Sinopsis: gserver

Éste es el servidor GSSAPI (Servicios Generales de Seguridad API). Este mandato no se ejecuta normalmente por usuarios. En vez de esto, se arranca en el lado del servidor cuando un usuario conecta desde un cliente con el método de acceso `:gserver:`:

```
cvs -d :gserver:floss.red-bean.com:/usr/local/nuevorepos checkout miproyecto
```

GSSAPI provee, entre otras cosas, la Versión 5 de Kerberos; para la versión 4, usar `:kserver:`.

Ajustando y usando una librería GSSAPI en sus máquinas está fuera del alcance de este libro. (Ver sin embargo el nodo *Acreditación GSSAPI* en el manual Cederqvist para obtener ayudas más útiles).

Opciones: ninguna.

history [OPCIONES] [SUBCADENA_FICHERO(S)]

Nombres alternativos – hi, his

Requiere – Repositorio, CVSROOT/history

Cambia – Nada

Muestra un historial de la actividad en el repositorio. Específicamente, esta opción muestra los registros de obtenciones, entregas, etiquetados, actualizaciones y lanzamientros.

Por omisión, la opción muestra las obtenciones (pero vea la opción -x). Este mandato no funciona si no hay fichero '`CVSROOT/history`' en el repositorio.

El mandato history difiere de otros mandatos CVS en muchos sentidos. Primero, deben proporcionarse opciones normalmente para hacer algo útil (y muchas de estas opciones significan diferentes cosas para el historial que para otros mandatos). Segundo, en vez de hablar de nombres completos de fichero como argumentos, toma una o más subcadenas para coincidir con nombres de fichero (todas las grabaciones que coincidan con al menos una de estas subcadenas se recuperan). Tercero, la salida del historial tiene aspecto de basura hasta que aprenda a leerla, así que explicaré el formato de salida en una sección especial después de las opciones. (Ver también ⟨undefined⟩ [log], page ⟨undefined⟩.)

Opciones:

-a – Muestra el historial para todos los usuarios (de otro modo, es por omisión uno mismo).

-b CADENA – Muestra los datos conteniendo la CADENA en el nombre del módulo, nombre del fichero o ruta al repositorio.

-c – Muestra entregas.

-D FECHA – Muestra los datos desde FECHA (los formatos de fecha de CVS están disponibles).

-e – Todo – Muestra todos los tipos de registros.

-f FICHERO – Muestra los eventos más recientes concernientes a FICHERO. Puede especificar esta opción muchas veces. Esto es diferente del significado usual de -f en los mandatos CVS: "Forzar a la revisión de cabecera como última solución."

-l – Muestra el registro representando el último (como en "más reciente") evento de cada proyecto. Esto es diferente del significado usual de -l en mandatos CVS: "Ejecutar localmente, no recursivamente".

-m MÓDULO – Esto produce un informe sobre el MÓDULO (nombre de proyecto). Puede especificar esta opción muchas veces.

-n MÓDULO – Muestra los eventos más recientes sobre MÓDULO. Por ejemplo, obtener el módulo concierne al módulo únicamente, pero modificar o actualizar un fichero dentro del módulo, trata del fichero, no del módulo. Puede especificar esta opción muchas veces. Ésto es diferente del significado usual de -n en los mandatos CVS: "No ejecutar un programa CVSROOT/modules."

-o – Muestra los registros de obtenciones (por omisión).

-p REPOS – Muestra los datos de un directorio particular en el repositorio. Puede especificar esta opción muchas veces. El significado de esta opción difiere del significado usual de -p en los mandatos CVS: "Encaminar los datos a la salida estándar en vez de hacia un fichero".

Esta opción parece estar por lo menos parcialmente rota desde el verano de 1999.

-r REV – Muestra los registros refirientes a versiones desde que la revisión o etiqueta llamada REV aparece en ficheros RCS individuales. En cada fichero RCS se busca la revisión o etiqueta.

-T – Muestra todos los eventos de etiquetado.

-t ETIQUETA – Muestra los registros desde que la ETIQUETA fue últimamente añadida al fichero de historial. Esto difiere del flag -r en donde se lee sólo el fichero '`CVSROOT/history`', no los ficheros RCS y es, por tanto, mucho más rápido.

-u USUARIO – Muestra los eventos asociados con USUARIO. Puede especificar esta opción muchas veces.

-w – Muestra los registros que están asociados con el directorio de trabajo del cual está invocando history.

-X FICHERO_HISTORIAL – Usa FICHERO_HISTORIAL en vez de '`CVSROOT/history`'. Esta opción es principalmente para depurado y no se apoya oficialmente; aun con todo, puede encontrarla útil (tal vez para generar informes legibles-por-humanos de ficheros viejos de historial que ha conservado por ahí).

-x TIPOS – Muestra eventos especificados en TIPOS. Cada tipo está representado por una simple letra, del juego '`TOEFWUCGMAR`'; cualquier número de letras puede ser combinado. Aquí está lo que significan:

T – Etiqueta

O – Obtención

E – Exportación

F – Lanzamiento

W – Actualización (fichero obsoleto más nuevo quitado de la copia de trabjo)

U – Actualización (fichero que fue entregado sobre un fichero de usuario)

C – Actualización (mezclado, con conflictos)

G – Actualización (mezclado, sin conflictos)

M – Entrega (el fichero fue modificado)

A – Entrega (el fichero fue añadido)

R – Entrega (el fichero fue borrado)

Por omisión, si no se proporcionan opciones -x, es para mostrar obtenciones (como `-x O`).

-z ZONA – Muestra tiempos en la salida para el huso ZONE. ZONE es un nombre de huso abreviado, como UTC, GMT, BST, CDT, CCT y así. Una lista completa de husos está disponible en TimezoneTable en el fichero '`lib/getdate.c`' en la distribución de fuentes de CVS.

Salida de historial

La salida del mandato history es una serie de líneas; cada línea representa un "evento de historial" y comienza con una sola letra de código indicando qué tipo de evento es. Por ejemplo:

```
floss$ cvs history -D yesterday -x TMO
M 08/21 20:19 +0000 jcualquiera 2.2           baar       miproyecto == <remote>
M 08/22 04:18 +0000 jcualquiera 1.2           README     miproyecto == <remote>
O 08/22 05:15 +0000 jcualquiera myproj =myproj= ~/src/*
M 08/22 05:33 +0000 jcualquiera 2.18          README.txt miproyecto == ~/src/myproj
O 08/22 14:25 CDT jcualquiera miproyecto =miproyecto= ~/src/*
O 08/22 14:26 CDT jcualquiera [99.08.23.19.26.03] miproyecto =miproyecto= ~/src/*
```

```
O 08/22 14:28 CDT jcualquiera [Saludos_Exoticos-rama] miproyecto =miproyecto= ~/src/*
```

Las letras de código son las mismas que para la opción -x recientemente descrita. Siguiendo el código de letra en la fecha de un evento (expresada en tiempo UTC/GMT, a no ser que se use la opción -z), seguida del usuario responsable del evento.

Después del usuario podría haber un número de revisión, etiqueta, o fecha, pero sólo si es apropiada para el evento (la fecha o etiqueta estarán en corchetes y formateados como se muestra en el ejemplo precedente). Si usted entrega un fichero, muestra el nuevo número de revisión; si obtiene con -D o -r, la fecha o etiqueta adhesiva se muestra dentro de corchetes. Para una obtención sencilla, nada extra se muestra.

Luego viene el nombre del fichero en cuestión, o nombre del módulo en el evento que es sobre un módulo. En el anterior, las siguientes dos cosas son el nombre de la copia de trabajo del módulo obtenido (entre dos signos de igual), seguida por su localización en el directorio personal del usuario (home). (El nombre de la copia de trabajo obtenida puede diferir del nombre del módulo si flag -d es usado con checkout).

import

Sinopsis: import [OPCIONES] REPOSITORIO ETIQUETA_VENDEDOR ETIQUETA_LANZAMIENTO

Nombres alternativos – im, imp

Requiere – Repositiorio, directorio actual (el directorio de fuentes)

Changes – Repositorio

Importa nuevas fuentes en el repositorio, bien creando un nuevo proyecto o creando una nueva revisión de vendedor en una rama de vendedor de un proyecto existente. (Ver ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩ para obtener una explicación básica de las ramas de vendedor en la importación, que le ayudará a entender lo siguiente).

Es normal usar import para añadir muchos ficheros o directorios de vez o para crear un nuevo proyecto. Para anñadir simples ficheros, debe usar add.

Opciones:

-b RAMA – Importa a la rama de vendedor RAMA. (RAMA es un número de rama actual, no una etiqueta). Esto se usa raramente pero puede ser útil si coge las fuentes del mismo proyecto de diferentes vendedores. Un mandato de importación normal asume que los fuentes deben ser importados en la rama de vendedor por omisión, que es "1.1.1". Debido a que es por omisión, normalmente no tiene que preocuparse en especificarlo con -b:

```
floss$ cvs import -m "importando del vendedor 1" su_proyecto ELLOS1 ELLOS1-0
```

Para importar a una rama de vendedor distinta de la que es por omisión, debe especificar un número de rama diferente explícitamente:

```
floss$ cvs import -b 1.1.3 -m "del vendedor 2" su_proyecto ELLOS2 ELLOS2-0
```

La rama 1.1.3 puede absorver futuras importaciones y ser mezclada como cualquier otra rama de vendedor. Sin embargo, debe asegurarse que en cualquier futura importación que especifique `-b 1.1.3` también debe usar la misma etiqueta de vendedor (`ELLOS2`). CVS no comprueba que el nombre de vendedor coincida con la etiqueta de vendedor. Sin embargo, si no coinciden, ocurrirán cosas extrañas e impredecibles.

Las ramas de vendedor están numeradas con impares, lo contrario de ramas regulares.

-d – Toma la fecha de modificación como la fecha de importación en vez de usar la fecha actual. Esto no funciona con CVS cliente/servidor.

-I NOMBRE – Da nombres de fichero que deben no ser tenidos en cuenta en la importación. Puede usar esta opción muchas veces en una importación. Los patrones con comodines están admitidos: `*.foo` significa no tener en cuenta todo lo que termine en '`.foo`'. (Ver ⟨undefined⟩ [cvsignore], page ⟨undefined⟩ en ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ para obtener detalles acerca de los comodines).

Los siguientes ficheros y nombres de directorios son obviados por omisión:

```
.
..
.#*
#*
,*
_$*
*~
*$
*.a
*.bak
*.BAK
*.elc
*.exe
*.ln
*.o
*.obj
*.olb
*.old
*.orig
*.rej
*.so
*.Z
.del-*
.make.state
.nse_depinfo
core
CVS
CVS.adm
cvslog.*
RCS
RCSLOG
SCCS
tags
TAGS
```

Puede suprimir que no se tengan en cuenta estos patrones de nombre de fichero, así como cualquiera especificado en '`.cvsignore`', '`CVSROOT/cvsignore`', y la variable de entorno `$CVSIGNORE`, usando `-I !`. Esto es,

```
floss$ cvs import -I ! -m "importando el universo" proyecto VENDEDOR VENDEDOR_0
```

importa todos los ficheros en el árbol de directorio actual, incluso los que serían normalmente desechados.

Usar `-I !` borra cualquier lista de ficheros a no tener en cuenta que haya sido creada hasta este punto, así que cualquier opción -I que venga después debería ser anulada, pero cualquiera que venga después todavía contará. Así,

```
floss$ cvs import -I ! -I LÉAME.txt -m "algunos msj" su_proyecto ELLOS ELLOS_0
```

no es lo mismo que

```
floss$ cvs import -I LÉAME.txt -I ! -m "algunos msj" su_proyecto ELLOS ELLOS_0
```

El primero obvia (no importa) LÉAME.txt, mientras que el último lo importa.

-k MODO – Configura la clave de modo de sustitución RCS por omisión para los ficheros importados. (Ver ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en este capítulo para obtener una lista de modo válidos).

-m MENSAJE – Registra MENSAJE como mensaje de registro de importación.

-W SPEC – Especifica filtros basados en nombres de fichero que deben ser usados efectivamente para la importación. Puede usar esta opción muchas veces. (Vea ⟨undefined⟩ [cvswrappers], page ⟨undefined⟩ en ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ para obtener detalles acerca de las especificaciones de cobertura).

init

Sinopsis: init NUEVO_REPOSITORIO

Nombres alternativos – Ninguno

Requiere – Localización para el nuevo repositorio

Crea – Repositorio

Crea un nuevo repositorio (que es, el raíz de un repositorio en el cual se almacenan muchos proyectos diferentes). Casi siempre deseará usar la opción global -d con esto, como en

```
floss$ cvs -d /usr/local/otro_nombre_de_repositorio init
```

porque, incluso si tiene la variable de entorno CVSROOT ajustada, está apuntando probablemente a un repositorio existente, que puede ser inservible e incluso peligroso en el contexto de este mandato. (Véase ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ para averiguar los pasos adicionales que se deberían tomar después de inicializar un nuevo repositorio).

Opciones: Ninguna.

kserver

Sinopsis: kserver

Este es un servidor Kerberos. (Si tiene la versión 4 de las librerías de Kerberos o anteriores – Versión 5 usa GSSAPI, ver ⟨undefined⟩ [gserver], page ⟨undefined⟩.) Este mandato normalmente no se ejecuta directamente por usuarios pero se ejecuta en el lado del servidor cuando un usuario conecta desde un cliente con el método de acceso `:kserver:`:

```
cvs -d :kserver:floss.red-bean.com:/usr/local/nuevorepos checkout miproyecto
```

Configurar y usar Kerberos en su máquina está fuera del alcance de este libro. (Sin embargo, vea *Acreditación Kerberos* en el manual Cederqvist para obtener ayuda más útil.)

Opciones: Ninguna.

log

Sinopsis: log [OPCIONES] [FICHEROS]

Nombres alternativos – lo, rlog

Requiere – Copia de trabajo, repositorio

Cambia – Nada

Muestra mensajes de registro para un proyecto, o para ficheros dentro de un proyecto. La salida de resitro no está precisamente en el mismo estilo que la salida de otros mandatos CVS, porque el registro se basa en un programa RCS más viejo (rlog). Su formato de salida da una cabecera, conteniendo varias piezas de información sobre el fichero no-específica-de-una-revisión, seguida de los mensajes de registro (arreglados por revisión). Cada revisión no muestra meramente el número de revisión y los mensajes de registro, sino también el autor y la fecha de el cambio y el número de líneas añadidas o borradas. Siempre imprimidas en UTC (GMT), no en fecha local.

Debido a que la salida de log es por fichero, una simple entrega que involucra múltiples ficheros puede no parecer conceptualmente como un cambio atómico. Sin embargo, si lee todos los mensajes de registro y fechas cuidadosamente, podrá reconstruir qué ocurrió. (Para obtener más información sobre una herramienta que puede reformatear salida de log de muchos ficheros de una forma más legible, vea ⟨undefined⟩ [cvs2cl – Genera ChangeLogs al estilo GNU], page ⟨undefined⟩ en ⟨undefined⟩ [Herramientas de terceros], page ⟨undefined⟩ para obtener más detalles). (Ver también ⟨undefined⟩ [history file], page ⟨undefined⟩).

Opciones:

Mientras lee las siguientes opciones de filtrado, puede no quedar completamente claro cómo se comportan cuando se combinan. Un descripción precisa del comportamiento del registro es la que toma la intersección de las revisiones seleccionadas por -d, -s y -w, cuando intersectan con la unión de las seleccionadas por -b y -r.

-b – Imprime información de registro acerca de la rama por omisión solamente (usualmente la rama más alta del tronco). Esto se hace usualmente para evitar la impresión de los mensajes de registro de ramas laterales de desarrollo.

-dFECHAS – Imprime información de registro para sólo las revisiones que coincidan con la fecha o rango de fechas dado por FECHAS, una lista separada por comas. Las fechas se pueden dar en cualquiera de los formatos usuales (ver ⟨undefined⟩ [Formatos de Fecha], page ⟨undefined⟩ anteriormente en esta sección) y puede ser combinado en rangos como a continuación:

FECHA1<FECHA2 – Selecciona las revisiones creadas entre FECHA1 y FECHA2. Si FECHA1 es más vieja que FECHA2, use en cambio >; de otro modo no se obtendrán mensajes de registro.

<FECHA FECHA> – Todas las fechas desde FECHA o más tempranas.

>FECHA FECHA< – Todas las revisiones de FECHA o más adelante.

FECHA – Simplemente selecciona la revisión simple más reciente desde FECHA o más temprana.

Puede usar `<=` y `>=` en vez de `<` y `>` para indicar un rango exclusivo (de otro modo, los rangos son inclusivos). Los rangos múltiples pueden ser separados por comas, por ejemplo

```
floss$ cvs log -d"1999-06-01<1999-07-01;1999-08-01<1999-09-01"
```

selecciona los mensajes de registro de revisiones entregadas en junio o agosto de 1999 (pasando julio). No puede haber espacio entre -d y sus argumentos.

-h – Imprime sólo la información de cabecera de cada fichero, que incluye el nombre del fichero, el directorio de trabajo, la revisión de cabecera, la rama por omisi\'n, la lista de acceso, los bloqueos, los nombres simbólicos (etiquetas) y la clave de modo de sustituci\'n por omisión. No se imprimen mensajes de registro.

-l – Local. Se ejecuta sólo para los ficheros en el directorio de trabajo actual.

-N – Omite la lista de nombres simbólicos (etiquetas) de la cabecera. Esto puede ser útil cuando su proyecto tiene muchas etiquetas que no está interesado en ver en los mensajes de registro.

-R – Imprime el nombre del fichero RCS en el repositorio.

Esto es diferente del significado usual de -R: "recursivo". No hay modo de contrarrestar un -l para este mandato, así que no ponga log -l en su '`.cvsrc`'.

-rREVS – Muestra información de registro para las revisiones especificadas en REVS, una lista separada por comas. REVS puede contener tanto números de revisión y etiquetas. Los rangos pueden ser especificados así:

REV1:REV2 – Revisiones desde REV1 a REV2 (deben estar en la misma rama).

:REV – Revisiones desde el comienzo de la rama de REV hasta, e incluyendo REV.

REV: – Revisiones desde REV hasta el final de la rama de REV.

RAMA – Todas las revisiones de esta rama, desde la raíz hasta la punta.

RAMA1:RAMA2 – Un rango de ramas – todas las revisiones de todas las ramas de este rango.

RAMA. – La última revisión (punta) de una RAMA.

Finalmente, un solo -r, sin argumento, significa seleccionar la última revisión en la rama por omisión (normalmente el tronco). No puede haber espacio entre -r y sus argumentos.

Si el argumento a -r es una lista, está separada por comas, no por punto y coma, como -d.

-sESTADOS – Selecciona las revisiones cuyos atributos de estado coincida con uno de los estados dados por ESTADOS, una lista separada por comas. No puede haber espacios entre -s y sus argumentos.

Si los argumentos a -s son una lista, están separado por comas, no por puntos y comas como -d.

-t – Como -h, pero también incluye la descripción del fichero (su mensaje de creación).

-wUSUARIOS – Selecciona las revisiones entregadas por usuarios cuyos nombres de usuario aparezcan en la lista de usuarios separada por comas. Un -w suelto sin USUARIOS significa tomar el nombre de usuario de la persona que ejecute cvs log.

Recuerde que cuando los sobrenombres de usuario se efectúan (ver sección ⟨undefined⟩ [El servidor de autentificacion de contrasen˜as], page ⟨undefined⟩ en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩), CVS registra el nombre de usuario CVS, no el nombre de usuario de sistema, en cada entrega. Puede no haber espacio entre -w y sus argumentos.

Si el argumento a -w es una lista, está separada por comas, no separada por puntos y comas como -d.

login

Sinopsis: login

Nombres alternativos – logon, lgn

Requiere – Repositorio

Cambia – fichero ˜/.cvspass

Establece contacto con un servidor CVS y confirma la información de acreditación para un repositorio en particular. Este mandato no afecta ni a la copia de trabajo ni al repositorio; simplemente confirma una clave (para usar con el método de acceso :pserver:) con un repositorio y almacena la clave para uso posterior en el fichero .cvspass en su directorio de trabajo. Mandatos futuros que accedan al mismo repositorio con el mismo nombre de usuario no requerirán que vuelva a ejecutar login, porque el CVS del lado del cliente consultará el fichero .cvspass para obtener la clave.

Si usa este mandato, debe especificar un repositorio usando el método de acceso pserver, como en esto

```
floss$ cvs -d :pserver:jcualquiera@floss.red-bean.com:/usr/local/nuevorepos
```

o configurando la variable de entorno CVSROOT.

Si cambia la clave en el lado del servidor, debe volver a ejecutar login.

Opciones: Ninguna.

logout

Sinopsis: logout

Nombres alternativos – None

Requiere – fichero ˜/.cvspass

Cambia – fichero ˜/.cvspass

Lo contrario que login – borra la clave para este repositorio de .cvspass.

Opciones: Ninguna.

pserver

Sinopsis: pserver

Nombres alternativos – Ninguno

Requiere – Repositorio

Cambia – Nada

Es el servidor de acreditación por clave. Este mandato normalmente no se ejecuta directamente por usuarios, pero se ejecuta desde '`/etc/inetd.conf`' en el servidor cuando un usuario se conecta desde un cliente con el método de acceso `:pserver:`. (Ver también los mandatos ⟨undefined⟩ [login], page ⟨undefined⟩ y ⟨undefined⟩ [logout], page ⟨undefined⟩, y el fichero '`.cvspass`' en la sección ⟨undefined⟩ [Ficheros de Control de Ejecucion], page ⟨undefined⟩ en este capítulo. Ver ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ para obtener más detalles acerca de configurar un servidor CVS de acreditación por clave).

Opciones: Ninguna.

rdiff

Sinopsis: rdiff [OPTIONS] PROJECTS

Nombres alternativos – patch, pa

Requiere – Repositorio

Cambia – Nada

Es como el mandato diff, excepto que opera directamente en el repositorio y, por tanto, no requiere copia de trabajo. Este mandato sirve para obtener las diferencias entre un lanzamiento y otro de su proyecto en un formato adecuado como entrada al programa patch (tal vez para que pueda distribuir ficheros parche a usuarios que quieran una actualización).

La operación del programa patch está fuera del alcance de este libro. Sin embargo, note que si el fichero de parche contiene diferencias para ficheros en subdirectorios, puede necesitar usar la opción -p de patch para permitir que aplique las diferencias correctamente. (Ver la documentación de patch para obtener más información acerca de esto). (Ver también ⟨undefined⟩ [diff], page ⟨undefined⟩).

Opciones:

-c – Imprime salida en el formato de contexto diff (por omisión).

-D FECHA o -D FECHA1 -D FECHA2 – Con una fecha, esto muestra las diferencias entre los ficheros desde FECHA hasta las revisiones de cabecera. Con dos fechas, muestra las diferencias entre las fechas.

-f – Fuerza el uso de una revisión de cabecera si no se ecuentran revisiones coincidentes para la opción -D o -r (de otro modo, rdiff no tendría en cuenta el fichero).

-l – Local. No desciende a los subdirectorios.

-R – Recursivo. Desciende a los subdirectorios (por omisión). Sólo tiene que especificar esta opción para contrarrestar un -l en su .cvsrc.

-r REV -r REV1 -r REV2 – Con una revisión, esto muestra las diferencias entre la revisión REV de los ficheros y la revisión de cabecera. Con dos, muestra las diferencias entre las revisiones.

-s – Muestra un sumario de diferencias. Esto muestra qué ficheros han sido añadidos, modificados o borrados, sin mostrar los cambios en sus contenidos. La salida tiene este aspecto:

```
floss$ cvs -Q rdiff -s -D 1999-08-20 mi_proyecto
File mi_proyecto/Cosa.txt is new; current revision 1.4
File mi_proyecto/LÉAME.txt changed from revision 2.1 to 2.20
File mi_proyecto/baar is new; current revision 2.3
```

-t – Muestra la diferencia entre las dos revisiones superiores de cada fichero. Esto es un atajo útil para determinar los cambios más recientes en un proyecto. Esta opción es incompatible con -D y -r.

-u – Imprime salida en formato unidiff. Las versiones más viejas del parche no pueden manejar formato unidiff; por tanto, no use -u si está intentando generar un fichero de parche distribuible – use -c en su lugar.

-V (En desuso) – CVS ahora da un error si intenta usar esta opción. La he incluido aquí sólo en caso de que vea algún script viejo intentando usarla.

release

Sinopsis: release [OPCIONES] DIRECTORIO

Nombres alternativos – re, rel

Requiere – Copia de trabajo

Cambia – Copia de trabajo, CVSROOT/history

Cancela una obtención (indica que la copia de trabajo y no está en uso). A diferencia de la mayoría de los mandatos CVS que operan en una copia de trabajo, esta no se invoca dentro de la copia de trabajo, sino directamente encima de ella (en su directorio padre). Debe configurar su variable de entorno CVSROOT o bien usar la opción global -d, ya que CVS no podrá encontrar el repositorio desde la copia de trabajo.

Usar release nunca es necesario. Debido a que CVS no hace bloqueo normalmente, puede simplemente borrar su copia de trabajo.

Sin embargo, si no ha entregado cambios en su copia de trabajo y quiere que la cesación del trabajo sea notificada en el fichero CVSROOT/history (ver mandato history), debe usar release. CVS primero comprueba cualquier cambio no entregado; si hay alguno, avisa y pregunta si continuar. Una vez que la copia de trabajo esté lanzada, este hecho se graba en el fichero del repositorio CVSROOT/history.

Opciones:

-d – Borra la copia de trabajo si el lanzamiento tiene éxito. Sin -d, la copia de trabajo permanece en el disco después del lanzamiento.

Si creó cualquier directorio nuevo dentro de su copia de trabajo pero no los añadió al repositorio, serán borrados junto con el resto de la copia de trabajo, si especificó la opción -d.

remove

Synopsis: remove [OPCIONES] [FICHEROS]

Nombres alternativos – rm, delete

Requiere – Copia de trabajo

Cambia – Copia de trabajo

Borra un fichero de un proyecto. Normalmente, el fichero en sí mismo ya está borrado del disco cuando invoca este mandato (pero vea -f). Aunque este mandato opera recursivamente por omisión, es común nombrar explícitamente el los ficheros que se van a borrar. Note lo

que implica la anterior afirmación: Usualmente, ejecuta cvs remove en los ficheros que no ya existen en su copia de trabajo.

Aunque se conecta con el repositorio para obtener confirmación, el fichero no se borra efectivamente hasta que una entrega posterior se realiza. Incluso entonces, el fichero RCS no se borra realmente de repositorio; si es borrado del tronco, se mueve al subdirectorio Attic/, donde todavía está disponible para su existencia en las ramas. Si se borra de una rama, su localización no se cambia, pero una nueva revisión con el estado dead (muerta) se añade a la rama (Ver también ⟨undefined⟩ [add], page ⟨undefined⟩).

Opciones:

-f – Fuerza. Borra el fichero de desco antes de borrarlo del CVS. Este significado difiere del usual significado de -f en los mandatos CVS: "Forzar a la revisión de cabecera".

-l – Local. Ejecuta sólo en el directorio de trabajo.

-R – Recursivo. Desciende dentro de los subdirectorios (por omisión). Esta opción existe sólo para contrarrestar un -l en .cvsrc.

rtag

Sinopsis: rtag [OPCIONES] TAG PROJECTO(S)

Nombres alternativos – rt, rfreeze

Requiere – Repositorio

Cambia – Repositorio

Etiqueta un módulo directamente en el repositorio (no requiere copia de trabajo). Probablemente necesita tener su variable de entorno CVSROOT configurada o usar la opción global -d para que esto funcione. (Ver también ⟨undefined⟩ [tag], page ⟨undefined⟩).

Opciones:

-a – Borra la etiqueta de cualquier fichero borrado, porque los ficheros borrados permanecen en el repositorio para propósitos de historial, pero ya no son considerados parte del proyecto vivo. Aunque es ilegal etiquetar ficheros con un nombre de etiqueta que ya está en uso, no debería interferir si el nombre sólo es usado en ficheros borrados (los cuales, desde el actual punto de vista del proyecto, ya no existen).

-b – Crea una nueva rama, con nombre de rama ETIQUETA.

-D FECHA – Etiqueta la última revisión no posterior a FECHA.

-d – Borra la etiqueta. No se hace ningún registro de este cambio – la etiqueta simplemente desaparece. CVS no mantiene un historial de cambios para las etiquetas.

-F – Fuerza la reasignación del nombre de la etiqueta, si ocurre que ya existe para alguna otra revisión del fichero.

-f – Fuerza a la revisión de cabecera si una etiqueta dada o fecha no es encontrada. (Ver -r y -D).

-l – Local. Ejecuta en el directorio actual solamente.

-n – No ejecuta un programa de etiquetación de CVSROOT/modules (Vea la sección ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ más adelante en este capítulo para obtener detalles acerca de estos programas).

-R – Recursivo. Desciende dentro de subdirectorios (por omisión). La opción -R existe sólo para contrarrestar un -l en un .cvsrc.

-r REV – Etiqueta la revisión REV (que puede ser en sí un nombre de etiqueta).

server

Sinopsis: server

Comienza un servidor CVS. Este mandato no se invoca nunca por usuarios (a no ser que esté intentando depurar el protocolo cliente/servidor), así que permítame limitarme a mencionarla.

Opcions: Ninguna.

status

Synopsis: status [OPCIONES] [FICHEROS]

Nombres alternativos – st, stat

Requiere – Copia de trabajo

Cambia – Nada

Muestra el estado de los ficheros en la copia de trabajo

Opciones:

-l – Local. Se ejecuta en el directorio actual solamente.

-R – Recursivo. Desciende dentro de los subdirectorios (por omisión). La opción -R existe sólo para contrarrestar un -l en .cvsrc.

-v – Muestra información de etiquetas para el fichero.

tag

Sinopsis: tag [OPCIONES] TAG [FICHEROS]

Nombres alternativos – ta, freeze

Requiere – Copia de trabajo, repositorio

Cambia – Repositorio

Asigna un nombre a una revisión particular o colección de revisiones para un proyecto. Muchas veces se llama "tomar una instantánea" del proyecto. Este mandato también se usa para crear ramas en CVS. (Ver la opción -b – ver también ⟨undefined⟩ [rtag], page ⟨undefined⟩).

Opciones:

-b – Crea una rama llamada TAG.

-c – Comprueba que la copia de trabajo no tenga cambios sin entregar. Si es así, el mandato sale con un aviso, y no se hace el etiquetado.

-D FECHA – Etiqueta la última revisión no posterior a FECHA.

-d – Borra la etiqueta. No se graba este cambio; la etiqueta simplemente desaparece. CVS no mantiene un historial de cambios de las etiquetas.

-F – Fuerza la reasignación del nombre de la etiqueta, si ocurre que existe ya para otras revisiones del fichero.

-f – Fuerza a la revisión de cabecera si una etiqueta dada o fecha no es encontrada. (Ver -r y -D).

-l – Local. Ejecuta sólo en el directorio actual.

-R – Recursivo. Desciende en los subdirectorios (por omisión). La opción -R existe sólo para contrarrestar un -l en un .cvsrc.

-r REV – Etiqueta la revisión REV (que puede ser en sí un nombre de etiqueta).

unedit

Sinopsis: unedit [OPCIONES] [FICHEROS]

Nombres alternativos – Ninguno

Requiere – Copia de trabajo, repositorio

Cambia – listas de edición/observación en el repositorio

Avisa a los observadores de que ha terminado de editar un fichero. (Ver también ⟨undefined⟩ [watch], page ⟨undefined⟩, ⟨undefined⟩ [watchers], page ⟨undefined⟩, ⟨undefined⟩ [edit], page ⟨undefined⟩ y ⟨undefined⟩ [editors], page ⟨undefined⟩).

Opciones:

-l – Local. Avisa sobre la edición para los ficheros del directorio de trabajo solamente.

-R – Recursivo (contrario de -l). Recursivo es por omisión; la única razón de pasar -R es contrarrestar un -l en su fichero .cvsrc.

update

Sinopsis: update [OPCIONES] [FICHEROS]

Nombres alternativos – up, upd

Requiere – Copia de trabajo, repositorio

Cambia – Copia de trabajo

Mezcla los cambios del repositorio en su copia de trabajo. Como efecto colateral, indica qué ficheros en su copia de trabajo están modificados (pero si la opción global -Q se pasa, estas indicaciones no se imprimen). (Véase también ⟨undefined⟩ [checkout], page ⟨undefined⟩).

Opciones:

-A – Borra cualquier etiqueta adhesiva, o cualquier clave RCS de modo de expansión adhesiva. Esto puede resultar en que los contenidos de los ficheros cambian, si las revisiones de la cabeza del tronco son diferentes del las anteriores revisiones adhesivas . (Imagine -A como si fuera una obtención fresca del tronco de un proyecto).

-C – Borra cualquier fichero localmente alterado y los remplaza con las últimas versiones del repositorio. Esto no es necesariamente lo mismo que revertir los ficheros, dado que el repositorio podría tener cambios desde la última actualización u obtención. Cualquier modificación se salva en '`.#fechero.rev`'.

Nota: Esta opción fue implementada en enero de 2000; si su CVS fue adquirido antes que entonces, tendrá que actualizarlo.

-D FECHA – Actualiza a la revisión más reciente no posterior a FECHA. Esta opción es adhesiva e implica -P. Si la copia de trabajo tiene fecha adhesiva, las entregas no son posibles.

-d – Recupera directorios ausentes – esto es, directorios que existen en el repositorio pero no todavía en la copia de trabajo. Tales directorios puden haber sido creados en el repositorio después de que la copia de trabajo fuera obtenida. Sin esta opción, update sólo opera en los directorios presentes en la copia de trabajo; los ficheros se traen desde el repositorio, pero los nuevos directorios no. (Ver también -P).

-f – Fuerza a la revisión de cabecera si no se encuentran revisiones coincidentes con las opciones -D o -r.

-I NOMBRE – Como la opción -I de import.

-j REV[:FECHA] o -j REV1[:FECHA] -j REV2[:FECHA] – Une, o mezcla, dos líneas de desarrollo. No teniendo en cuenta el argumento opcional de FECHA por el momento (lo retomaremos luego), así es cómo -j funciona: Si sólo se da una, toma todos los cambios desde el ancestro común a REV y los mezcla en la copia de trabajo. El *acestro común* es la última revisión que es acestral a ambas revisiones en el directorio de trabajo y a REV. Si se dan dos opciones -j, mezcla los cambios de REV1 a REV2 en la copia de trabajo.

Las etiquetas especiales HEAD y BASE pueden ser usadas como argumentos de -j; significan la más reciente revisión del repositorio y la revisión en la que se basa la copia actual, respectivamente.

Y para los argumentos opcionales de FECHA, si REV es una rama, se toma normalmente el significado de la última revisión no posterior a FECHA. La fecha debe estar separada de la revisión por una coma, sin espacios, por ejemplo:

```
floss$ cvs update -j UnaRama:1999-07-01 -j UnaRama:1999-08-01
```

En este ejemplo, diferentes fechas en la misma rama se usan, así el efecto es tomar los cambios en esta rama desde julio a agosto y mezclarlos en la copia de trabajo. Sin embargo, note que no se requiere que la rama sea la misma en ambas opciones -j.

-k MODO – Realiza una sustitución RCS de acuerdo al MODO. (Vea la sección ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ más adelante en este capítulo). El modo permanece adhesivo en la copia de trabajo, así que afectará a futuras actualizaciones (pero vea -A).

-l – Local. Actualiza sólo el directorio actual.

-P – Poda los directorios vacíos. Cualquier directorio controlado por CVS que no contenga ficheros al final de la actualización se borra de la copia de trabajo. (Ver también -d).

-p – Envía los contenidos del fichero a la salida estándar en vez de a los ficheros. Usando normalmente para revertir a una revisión anterior sin producir etiquetas adhesivas en la copia de trabajo. Por ejemplo:

```
floss$ cvs update -p -r 1.3 LÉAME.txt > LÉAME.txt
```

Ahora LÉAME.txt en la copia de trabajo tiene los contenidos de su pasada revisión 1.3, simplemente como si lo tuviera editado a mano en este estado.

-R – Recursivo. Desciende a los subdirectorios para actualizar (por omisión). La única razón por la que lo especificaría sería para contrarrestar un -l en .cvsrc.

-r REV – Actualiza (o rejuvenece) a la revisión REV. Cuando actualiza una copia de trabajo entera, REV es habitualmente una etiqueta (regular o rama). Sin embargo, cuando actualice un fichero individual, es tan probable que sea un número de revisión como una etiqueta.

Esta opción es adhesiva. Si los ficheros se cambian a una etiqueta que no sea de una rama o revisiones adhesivas, no pueden ser omitidas hasta que lo adhesivo se quite. (Véase -A). Si REV es una etiqueta de rama, sin embargo, las entrgas son posibles. Simplemente se entregarán nuevas revisiones en esta rama.

-WSPEC – Especifica filtros de cobertura para usarse durante la actualización. Puede usar esta opción muchas veces. (Vea ⟨undefined⟩ [cvswrappers], page ⟨undefined⟩ en ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ en este capítulo para obtener más detalles sobre las especificaciones de cobertura). No hay espacio entre -W y su argumento.

watch

Sinopsis: watch on | off | add | remove [OPCIONES] [FICHEROS]

Nombres alternativos – Ninguno

Requiere – Copia de trabajo, repositorio

Cambia – Lista de observaciones en el repositorio

Configura en observación en uno o más ficheros. A diferencia de la mayoría de los mandatos de CVS, la observación requiere un submandato adicional para hacer algo útil. (Véase también ⟨undefined⟩ [watchers], page ⟨undefined⟩, ⟨undefined⟩ [edit], page ⟨undefined⟩, ⟨undefined⟩ [editors], page ⟨undefined⟩, ⟨undefined⟩ [unedit], page ⟨undefined⟩ y ⟨undefined⟩ [users], page ⟨undefined⟩).

Submandatos:

on – Declara que los ficheros están siendo observados. Esto significa que se crean en modo de sólo-lectura en la obtención, y los usuarios deben hacer cvs edit para crearlos como lectura-escritura (notificar a cualquier observador que el fichero ahora está siendo editado). Conectar una observación no le añade a la lista de observadores para ningún fichero. (Vea `watch add` y `watch remove` para eso).

off – Contrario de watch on. Declara que los ficheros ya no están siendo observados.

add – Le añade a la lista de observadores para este fichero. Usted será notificado cuando alguien entregue o ejecute cvs edit o cvs unedit (pero vea la opción -a).

remove – Contrario de watch add. Le borra de la lista de observadores para este fichero.

Opciones (para usar cualquier submandato de observación). Las tres funciones tienen los mismos significados que para editar:

-a ACCIONES

-l

-R

watchers

Sinopsis: watchers [OPCIONES] [FICHEROS]

Nombres alternativos – Ninguno

Requiere – Copia de trabajo, repositorio

Cambia – Nada

Muestra quién está observando qué ficheros.

Opciones – estas opciones significan la misma cosa aquí que para ⟨undefined⟩ [edit], page ⟨undefined⟩:

-l

-R

Claves de Sustitucion (Claves RCS)

CVS puede realizar algunas sustituciones en ficheros, permitiéndole que mantenga automáticamente alguna información actualizada en sus ficheros. Todas las sustituciones se realizan a través de unos patrones de palabras claves determinadas, rodeadas por los símbolos de dolar. Por ejemplo:

```
$Revision: 1.6 $
```

en un fichero se expande a algo como

```
$Revision: 1.6 $
```

y CVS seguirá manteniendo la cadena de revisión actualizada a medida que se añaden nuevas revisiones.

Controlling Keyword Expansion

By default, CVS performs keyword expansion unless you tell it to stop. You can permanently suppress keyword expansion for a file with the -k option when you add the file to the project, or you can turn it off later by invoking admin with -k. The -k option offers several different modes of keyword control; usually you want mode o or b, for example:

```
floss$ cvs add -ko chapter-9.sgml
```

This command added '`chapter-9.sgml`' to the project with keyword expansion turned off. It sets the file's default keyword expansion mode to `o`, which means no substitution. (Actually, the "o" stands for "old", meaning to substitute the string with its old value, which is the same as substituting it for itself, resulting in no change. I'm sure this logic made sense to somebody at the time.)

Each file's default keyword mode is stored in the repository. However, each working copy can also have its own local keyword substitution mode – accomplished with the -k options to checkout or update. You can also have a mode in effect for the duration of just one command, with the -k option to diff.

Here are all the possible modes, presented with the -k option prepended (as one would type at a command line). Any of these options can be used as either the default or local keyword substitution mode for a file:

-kkv – Expands to keyword and value. This is the default keyword expansion mode, so you don't need to set it for new files. You might use it to change a file from another keyword mode, however.

-kkvl – Like -kkv, but includes the locker's name if the revision is currently locked. (See the -l option to admin for more on this.)

-kk – Won't expand values in keyword strings, just uses the keyword name. For example, with this option,

```
$Revision: 1.6 $
```

and

```
$Revision: 1.6 $
```

would both "expand" (okay, contract) to:

```
$Revision: 1.6 $
```

-ko – Reuses the keyword string found in the file (hence "o" for "old"), as it was in the working file just before the commit.

-kb – Like -ko, but also suppresses interplatform line-end conversions. The "b" stands for "binary"; it is the mode you should use for binary files.

-kv – Substitutes the keyword with its value, for example

```
$Revision: 1.6 $
```

might become:

```
1.5
```

Of course, after that's happened once, future substitutions will not take place, so this option should be used with care.

List Of Keywords

These are all the dollar-sign-delimited keywords that CVS recognizes. Following is a list of the keyword, a brief description, and an example of its expanded form:

$Author: jfs $ – Author of the change:

```
$Author: jfs $
```

$Date: 2002/12/05 19:10:27 $ – The date and time of the change, in UTC (GMT):

```
$Date: 2002/12/05 19:10:27 $
```

$Header: /home/cvs/lucas/doc-cvsbook-es/chapter-9.texi,v 1.6 2002/12/05 19:10:27 jfs Exp $ – Various pieces of information thought to be useful: full path to the RCS file in the repository, revision, date (in UTC), author, state, and locker. (Lockers are rare; although in the following example, qsmith has a lock.):

```
$Header: /usr/local/newrepos/myproj/hello.c,v 1.1 1999/06/01 \
03:21:13 jrandom Exp qsmith $
```

$Id: chapter-9.texi,v 1.6 2002/12/05 19:10:27 jfs Exp $ – Like $Header: /home/cvs/lucas/doc-cvsbook-es/chapter-9.texi,v 1.6 2002/12/05 19:10:27 jfs Exp $, but without the full path to the RCS file:

```
$Id: chapter-9.texi,v 1.6 2002/12/05 19:10:27 jfs Exp $
```

$Log: chapter-9.texi,v $ Revision 1.6 2002/12/05 19:10:27 jfs Ahora el documento compila incluyendo los acentos en castellano, he seguido los consejos de http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html Sin embargo sería conveniente revisar por qué los nombres de nodo que se referencian no pueden estar con ISO-latin1... problema de TexInfo?

Revision 1.5 2002/12/05 17:12:19 jfs Actualizado el LEAME y traducido (un poco) el capítulo 9

Revision 1.4 2002/12/03 13:04:59 carlosgarcia Traduccion de main, introduction e index

Revision 1.3 2002/11/27 16:25:32 carlosgarcia Falta traducir el final del fichero – The log message of this revision, along with the revision number, date, and author. Unlike other keywords, the previous expansions are not replaced. Instead, they are pushed down, so that the newest expansion appears at the top of an ever-growing stack of $Log: chapter-9.texi,v $ newest expansion appears at the top of an ever-growing stack of Revision 1.6 2002/12/05 19:10:27 jfs newest expansion appears at the top of an ever-growing stack of Ahora el documento compila incluyendo los acentos en castellano, he seguido newest expansion appears at the top of an ever-growing stack of los consejos de newest expansion appears at the top of an ever-growing stack of http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html newest expansion appears at the top of an ever-growing stack of Sin embargo sería conveniente revisar por qué los nombres de nodo que se newest expansion appears at the top of an ever-growing stack of referencian no pueden estar con ISO-latin1... problema de TexInfo? newest expansion appears at the top of an ever-growing stack of newest expansion appears at the top of an ever-growing stack of Revision 1.5 2002/12/05 17:12:19 jfs newest expansion appears at the top of an ever-growing stack of Actualizado el LEAME y traducido (un poco) el capítulo 9 newest expansion appears at the top of an ever-growing stack of newest expansion appears at the top of an ever-growing stack of Revision 1.4 2002/12/03 13:04:59 carlosgarcia newest expansion appears at the top of an ever-growing stack of Traduccion de main, introduction e index newest expansion appears at the top of an ever-growing stack of newest expansion appears at the top of an ever-growing stack of Revision 1.3 2002/11/27 16:25:32 carlosgarcia newest expansion appears at the top of an ever-growing stack of Falta traducir el final del fichero newest expansion appears at the top of an ever-growing stack of messages:

```
$Log: chapter-9.texi,v $
Revision 1.6  2002/12/05 19:10:27  jfs
Ahora el documento compila incluyendo los acentos en castellano, he seguido
los consejos de
http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html
Sin embargo sería conveniente revisar por qué los nombres de nodo que se
referencian no pueden estar con ISO-latin1... problema de TexInfo?

Revision 1.5  2002/12/05 17:12:19  jfs
Actualizado el LEAME y traducido (un poco) el capítulo 9

Revision 1.4  2002/12/03 13:04:59  carlosgarcia
Traduccion de main, introduction e index

Revision 1.3  2002/11/27 16:25:32  carlosgarcia
```

```
Falta traducir el final del fichero
    Revision 1.12  1999/07/19 06:12:43  jrandom
    say hello in Aramaic
```

Any text preceding the $Log: chapter-9.texi,v $ Any text preceding the Revision 1.6 2002/12/05 19:10:27 jfs Any text preceding the Ahora el documento compila incluyendo los acentos en castellano, he seguido Any text preceding the los consejos de Any text preceding the http://www.geocities.com/sl_edu_colombia/soluciones/vladimir/linux_esp.html Any text preceding the Sin embargo sería conveniente revisar por qué los nombres de nodo que se Any text preceding the referencian no pueden estar con ISO-latin1... problema de TexInfo? Any text preceding the Any text preceding the Revision 1.5 2002/12/05 17:12:19 jfs Any text preceding the Actualizado el LEAME y traducido (un poco) el capítulo 9 Any text preceding the Any text preceding the Revision 1.4 2002/12/03 13:04:59 carlosgarcia Any text preceding the Traduccion de main, introduction e index Any text preceding the Any text preceding the Revision 1.3 2002/11/27 16:25:32 carlosgarcia Any text preceding the Falta traducir el final del fichero Any text preceding the keyword on the same line will be prepended to the downward expansions too; this is so that if you use it in a comment in a program source file, all of the expansion is commented, too.

$Locker: $ – Name of the person who has a lock on this revision (usually no one):

```
$Locker:  $
```

$Name: $ – Name of the sticky tag:

```
$Name:  $
```

$RCSfile: chapter-9.texi,v $ – Name of the RCS file in the repository:

```
$RCSfile: chapter-9.texi,v $
```

$Revision: 1.6 $ – Revision number:

```
$Revision: 1.6 $
```

$Source: /home/cvs/lucas/doc-cvsbook-es/chapter-9.texi,v $ – Full path to the RCS file in the repository:

```
$Source: /home/cvs/lucas/doc-cvsbook-es/chapter-9.texi,v $
```

$State: Exp $ – State of this revision:

```
$State: Exp $
```

Ficheros de Administracion del Repositorio

The repository's administrative files are stored in the CVSROOT subdirectory of the repository. These files control various aspects of CVS's behavior (in that repository only, of course).

You may also want to refer to the discussion of administrative files in ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩, which includes examples.

Storage And Editing

Generally, the administrative files are kept under revision control just like any other file in the repository (the exceptions are noted). However, unlike other files, checked-out copies of

the administrative files are stored in the repository, right next to their corresponding RCS files in the '`CVSROOT`' subdirectory. It is these checked-out copies which actually govern CVS's behavior.

The normal way to modify the administrative files is to check out a working copy of the CVSROOT module, make your changes, and commit. CVS updates the checked-out copies in the repository automatically. (See ⟨undefined⟩ [checkoutlist], page ⟨undefined⟩.) In an emergency, however, it is also possible to edit the checked-out copies in the repository directly.

Shared Syntax

In all of the administrative files, a `#` at the beginning of a line signifies a comment; that line is ignored by CVS. A backslash preceding a newline quotes the newline out of existence.

Some of the files (commitinfo, loginfo, taginfo, and rcsinfo) share more syntactic conventions as well. In these files, on the left of each line is a regular expression (which is matched against a file or directory name), and the rest of the line is a program, possibly with arguments, which is invoked if something is done to a file matching the regular expression. The program is run with its working directory set to the top of the repository.

In these files, there are two special regular expressions that may be used: ALL and DEFAULT. ALL matches any file or directory, whether or not there is some other match for it, and DEFAULT matches only if nothing else matched.

Shared Variables

The info files also allow certain variables to be expanded at runtime. To expand a variable, precede it with a dollar sign (and put it in curly braces just to be safe). Here are the variables CVS knows about:

${CVSROOT} – The top of the repository.

${RCSBIN} – (Obsolete) Don't use this variable. It is only applicable in CVS Version 1.9.18 and older. Specifying it now may result in an error.

${CVSEDITOR} ${VISUAL} ${EDITOR} – These all expand to the editor that CVS is using for a log message.

${USER} – The user running CVS (on the server side).

User Variables

Users can also set their own variables when they run any CVS command. (See the -s global option.) These variables can be accessed in the '`*info`' files by preceding them with an equal sign, as in ${=VAR}.

checkoutlist

This contains a list of files for which checked-out copies should be kept in the repository. Each line gives the file name and an error message for CVS to print if, for some reason, the file cannot be checked out in the repository:

```
FILENAME  ERROR_MESSAGE
```

Because CVS already knows to keep checked-out copies of the existing administrative files, they do not need to be listed in checkoutlist. Specifically, the following files never need entries in checkoutlist: loginfo, rcsinfo, editinfo, verifymsg, commitinfo, taginfo, ignore, checkoutlist, cvswrappers, notify, modules, readers, writers, and config.

commitinfo

Specifies programs to run at commit time, based on what's being committed. Each line consists of a regular expression followed by a command template:

```
REGULAR_EXPRESSION PROGRAM [ARGUMENTS]
```

The PROGRAM is passed additional arguments following any arguments you may have written into the template. These additional arguments are the full path to the repository, followed by the name of each file about to be committed. These files can be examined by PROGRAM; their contents are the same as those of the working copy files about to be committed. If PROGRAM exits with nonzero status, the commit fails; otherwise, it succeeds. (See also ⟨undefined⟩ [Shared Syntax], page ⟨undefined⟩ earlier in this chapter.)

config

Controls various global (non-project-specific) repository parameters. The syntax of each line is

```
ParameterName=yes|no
```

except for the LockDir parameter, which takes an absolute pathname as argument.

The following parameters are supported:

RCSBIN (default: `=no`) – (Obsolete) This option is silently accepted for backwards compatibility, but no longer has any effect.

SystemAuth (default: `=no`) – If `yes`, CVS pserver authentication tries the system user database – usually '`/etc/passwd`' – if a username is not found in '`CVSROOT/passwd`'. If `no`, the user must exist in '`CVSROOT/passwd`' to gain access via the `:pserver:` method.

PreservePermissions (default: `=no`) – If `yes`, CVS tries to preserve permissions and other special file system information (such as device numbers and symbolic link targets) for files. You probably don't want to do this, as it does not necessarily behave as expected. (See the node *Special Files* in the Cederqvist manual for details.)

TopLevelAdmin (default: `=no`) – If `yes`, checkouts create a '`CVS/`' subdirectory next to each working copy tree (in the parent directory of the working copy). This can be useful if you will be checking out many working copies from the same repository; on the other hand, setting it here affects everyone who uses this repository.

LockDir (unset by default) – The argument after the equal sign is a path to a directory in which CVS can create lockfiles. If not set, lockfiles are created in the repository, in locations corresponding to each project's RCS files. This means that users of those projects must have file-system-level write access to those repository directories.

cvsignore

Ignores certain files when doing updates, imports, or releases. By default, CVS already ignores some kinds of files. (For a full list, see the -I option to import, earlier in this chapter.) You can add to this list by putting additional file names or wildcard patterns in the cvsignore file. Each line gives a file name or pattern, for example:

```
README.msdos
*.html
blah?.out
```

This causes CVS to ignore any file named '`README.msdos`', any file ending in '`.html`', and any file beginning with '`blah`' and ending with '`.out`'. (Technically, you can name multiple files or patterns on each line, separated by whitespace, but it is more readable to keep them to one per line. The whitespace separation rule does, unfortunately, mean that there's no way to specify a space in a file name, except to use wildcards.)

A `!` anywhere in the list cancels all previous entries. (See ⟨undefined⟩ [$CVSIGNORE], page ⟨undefined⟩ in the section ⟨undefined⟩ [Environment Variables], page ⟨undefined⟩ in this chapter for a fuller discussion of ignore processing.)

cvswrappers

Specifies certain filtering behaviors based on file name. Each line has a file-globbing pattern (that is, a file name or file wildcards), followed by an option indicating the filter type and an argument for the option.

Options:

-m – Specifies an update method. Possible arguments are MERGE, which means to merge changes into working files automatically, and COPY, which means don't try to automerge but present the user with both versions of the file and let them work it out. MERGE is the default, except for binary files (those whose keyword substitution mode is -kb). (See the ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ section in this chapter.) Files marked as binary automatically use the COPY method, so there is no need to make a -m COPY wrapper for them.

-k – Specifies a keyword substitution mode. All of the usual modes are possible. (See the ⟨undefined⟩ [Claves de Sustitucion (Claves RCS)], page ⟨undefined⟩ section in this chapter for a complete list.)

Here is an example cvswrappers file:

```
*.blob    -m COPY
*.blink   -k o
```

This cvswrappers file says to not attempt merges on files ending in '`.blob`' and suppress keyword substitution for files ending in '`.blink`'. (See also the file '`.cvswrappers`' in the ⟨undefined⟩ [Working Copy Files], page ⟨undefined⟩ section in this chapter.)

editinfo

This file is obsolete. Very.

history file

Stores an ever-accumulating history of activity in the repository, for use by the cvs history command. To disable this feature, simply remove the history file. If you don't remove the file, you should probably make it world-writeable to avoid permission problems later.

The contents of this file do not modify CVS's behavior in any way (except for the output of cvs history, of course).

loginfo

Specifies programs to run on the log message for each commit, based on what's being committed. Each line consists of a regular expression followed by a command template:

```
REGULAR_EXPRESSION PROGRAM [ARGUMENTS]
```

The PROGRAM is passed the log message on its standard input.

Several special codes are available for use in the arguments: `%s` expands to the names of the files being committed, `%V` expands to the old revisions from before the commit, and `%v` expands to the new revisions after the commit. When there are multiple files involved, each element of the expansion is separated from the others by whitespace. For example, in a commit involving two files, `%s` might expand into `hello.c README.txt`, and `%v` into `1.17 1.12`.

You may combine codes inside curly braces, in which case, each unit of expansion is internally separated by commas and externally separated from the other units by whitespace. Continuing the previous example, `%{sv}` expands into `hello.c,1.17 README.txt,1.12`.

If any `%` expansion is done at all, the expansion is prefixed by the path to the project subdirectory (relative to the top of the repository). So that last expansion would actually be:

```
myproj  hello.c,1.17  README.txt,1.12
```

If PROGRAM exits with nonzero status, the commit fails; otherwise, it succeeds. (See also the ⟨undefined⟩ [Shared Syntax], page ⟨undefined⟩ section in this chapter.)

modules

This maps names to repository directories. The general syntax of each line is:

```
MODULE [OPTIONS] [&OTHERMODULE...] [DIR] [FILES]
```

DIR need not be a top-level project directory – it could be a subdirectory. If any FILES are specified, the module consists of only those files from the directory.

An ampersand followed by a module name means to include the expansion of that module's line in place.

Options:

-a – This is an *alias* module, meaning it expands literally to everything after the OPTIONS. In this case, the usual DIR/FILES behavior is turned off, and everything after the OPTIONS is treated as other modules or repository directories.

If you use the -a option, you may exclude certain directories from other modules by putting them after an exclamation point (!). For example

```
top_proj -a !myproj/a-subdir !myproj/b-subdir myproj
```

means that checking out `top_proj` will get all of `myproj` except `a-subdir` and `b-subdir`.

-d NAME – Names the working directory NAME instead of the module name.

-e PROGRAM – Runs PROGRAM whenever files in this module are exported.

-i PROGRAM – Runs PROGRAM whenever files in this module are committed. The program is given a single argument – the full pathname in the repository of the file in question. (See ⟨undefined⟩ [commitinfo], page ⟨undefined⟩, ⟨undefined⟩ [loginfo], page ⟨undefined⟩, and ⟨undefined⟩ [verifymsg], page ⟨undefined⟩ for more sophisticated ways to run commit-triggered programs.)

-o PROGRAM – Runs PROGRAM whenever files in this module are checked out. The program is given a single argument, the name of the module.

-s STATUS – Declares a status for the module. When the modules file is printed (with cvs checkout -s), the modules are sorted by module status and then by name. This option has no other effects in CVS, so go wild. You can use it to sort anything – status, person responsible for the module, or the module's file language, for example.

-t PROGRAM – Runs PROGRAM whenever files in this module are tagged with cvs rtag. The program is passed two arguments: the name of the module and the tag name. The program is not used for tag, only for rtag. I have no idea why this distinction is made. You may find the taginfo file more useful if you want to run programs at tag time.

-u PROGRAM – Runs PROGRAM whenever a working copy of the module is updated from its top-level directory. The program is given a single argument, the full path to the module's repository.

notify

Controls how the notifications for watched files are performed. (You may want to read up on the watch and edit commands, or see the section ⟨undefined⟩ [Alarmas (CVS como telefono)], page ⟨undefined⟩ in ⟨undefined⟩ [CVS avanzado], page ⟨undefined⟩.) Each line is of the usual form:

REGULAR_EXPRESSION PROGRAM [ARGUMENTS]

A `%s` in ARGUMENTS is expanded to the name of the user to be notified, and the rest of the information regarding the notification is passed to PROGRAM on standard input (usually this information is a brief message suitable for emailing to the user). (See the section ⟨undefined⟩ [Shared Syntax], page ⟨undefined⟩ earlier in this chapter.)

As shipped with CVS, the notify file has one line

```
ALL mail %s -s "CVS notification"
```

which is often all you need.

passwd

Provides authentication information for the pserver access method. Each line is of the form:

USER:ENCRYPTED_PASSWORD[:SYSTEM_USER]

If no SYSTEM_USER is given, USER is taken as the system username.

rcsinfo

Specifies a form that should be filled out for log messages that are written with an interactive editor. Each line of rcsinfo looks like:

REGULAR_EXPRESSION FILE_CONTAINING_TEMPLATE

This template is brought to remote working copies at checkout time, so if the template file or rcsinfo file changes after checkout, the remote copies won't know about it and will continue to use the old template. (See also the section ⟨undefined⟩ [Shared Syntax], page ⟨undefined⟩ in this chapter.)

taginfo

Runs a program at tag time (usually done to check that the tag name matches some pattern). Each line is of the form:

REGULAR_EXPRESSION PROGRAM

The program is handed a set group of arguments. In order, they are the tag name, the operation (see below), the repository, and then as many file name/revision-number pairs as there are files involved in the tag. The file/revision pairs are separated by whitespace, like the rest of the arguments.

The operation is one of `add`, `mov`, or `del` (`mov` means the -F option to tag was used).

If PROGRAM exits with nonzero status, the tag operation will not succeed. (See also the section ⟨undefined⟩ [Shared Syntax], page ⟨undefined⟩ in this chapter.)

users

Maps usernames to email addresses. Each line looks like:

USERNAME:EMAIL_ADDRESS

This sends watch notifications to EMAIL_ADDRESS instead of to USERNAME at the repository machine. (All this really does is control the expansion of %s in the notify file.) If EMAIL_ADDRESS includes whitespace, make sure to surround it with quotes.

If user aliasing is being used in the passwd file, the username that will be matched is the CVS username (the one on the left), not the system username (the one on the right, if any).

val-tags

Caches valid tag names for speedier lookups. You should never need to edit this file, but you may need to change its permissions, or even ownership, if people are having trouble retrieving or creating tags.

verifymsg

Used in conjunction with rcsinfo to verify the format of log messages. Each line is of the form:

REGULAR_EXPRESSION PROGRAM [ARGUMENTS]

The full path to the current log message template (see ⟨undefined⟩ [rcsinfo], page ⟨undefined⟩ earlier in this chapter) is appended after the last argument written in the verifymsg file. If PROGRAM exits with nonzero status, the commit fails.

Ficheros de Control de Ejecucion

There are a few files on the client (working copy) side that affect CVS's behavior. In some cases, they are analogs of repository administrative files; in other cases, they control behaviors that are only appropriate for the client side.

'.cvsrc'

Specifies options that you want to be used automatically with every CVS command. The format of each line is

COMMAND OPTIONS

where each COMMAND is an unabbreviated CVS command, such as checkout or update (but not co or up). The OPTIONS are those that you want to always be in effect when you run that command. Here is a common '`.cvsrc`' line:

```
update -d -P
```

To specify global options, simple use cvs as the COMMAND.

'.cvsignore'

Specifies additional ignore patterns. (See ⟨undefined⟩ [cvsignore], page ⟨undefined⟩ in the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter for the syntax.)

You can have a .cvsignore file in your home directory, which will apply every time you use CVS. You can also have directory-specific ones in each project directory of a working copy (these last only apply to the directory where the .cvsignore is located, and not to its subdirectories).

(See ⟨undefined⟩ [$CVSIGNORE], page ⟨undefined⟩ in the section ⟨undefined⟩ [Environment Variables], page ⟨undefined⟩ in this chapter, for a fuller discussion of ignore processing.)

'.cvspass'

Stores passwords for each repository accessed via the pserver method. Each line is of the form:

REPOSITORY LIGHTLY_SCRAMBLED_PASSWORD

The password is essentially stored in cleartext – a very mild scrambling is done to prevent accidental compromises (such as the root user unintentionally looking inside the

file). However, this scrambling will not deter any serious-minded person from gaining the password if they get access to the file.

The .cvspass file is portable. You can copy it from one machine to another and have all of your passwords at the new machine, without ever having run cvs login there. (See also the ⟨undefined⟩ [login], page ⟨undefined⟩ and ⟨undefined⟩ [logout], page ⟨undefined⟩ commands.)

'.cvswrappers'

This is a client side version of the cvswrappers file. (See the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter.) There can be a '`.cvswrappers`' file in your home directory and in each directory of a working copy directory, just as with '`.cvsignore`'.

Working Copy Files

The CVS/ administrative subdirectories in each working copy contain some subset of the following files.

- CVS/Base/
- CVS/Baserev
- CVS/Baserev.tmp
- CVS/Checkin.prog
- CVS/Entries
- CVS/Entries.Backup
- CVS/Entries.Log
- CVS/Entries.Static
- CVS/Notify
- CVS/Notify.tmp
- CVS/Repository
- CVS/Root
- CVS/Tag
- CVS/Template
- CVS/Update.prog

Here is what each file or directory does:

'CVS/Base/' (directory)

If watches are on, `cvs edit` stores the original copy of the file in this directory. That way, `cvs unedit` can work even if it can't reach the server.

'CVS/Baserev'

Lists the revision for each file in '`Base/`'. Each line looks like this:

```
FILE/REVISION/EXPANSION
```

EXPANSION is currently ignored to allow for, well, future expansion.

'CVS/Baserev.tmp'

This is the temp file for the preceding. (See '`CVS/Notify.tmp`' or '`CVS/Entries.Backup`' later on for further explanation.)

'CVS/Checkin.prog'

Records the name of the program specified by the -i option in the modules file. (See the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter.)

'CVS/Entries'

Stores the revisions for the files in this directory. Each line is of the form:

```
[CODE_LETTER]/FILE/REVISION/DATE/[KEYWORD_MODE]/[STICKY_OPTION]
```

If CODE_LETTER is present, it must be `D` for directory (anything else is silently ignored by CVS, to allow for future expansion), and the rest of the items on the line are absent.

This file is always present.

'CVS/Entries.Backup'

This is just a temp file. If you're writing some program to modify the '`Entries`' file, have it write the new contents to '`Entries.backup`' and then atomically rename it to '`Entries`'.

'CVS/Entries.Log'

This is basically a patch file to be applied to '`Entries`' after '`Entries`' has been read (this is an efficiency hack, to avoid having to rewrite all of '`Entries`' for every little change). The format is the same as '`Entries`', except that there is an additional mandatory code letter at the front of every line: An `A` means this line is to be added to what's in '`Entries`'; `R` means it's to be removed from what's in '`Entries`'. Any other letters should be silently ignored, to allow for future expansion.

'CVS/Entries.Static'

If this file exists, it means only part of the directory was fetched from the repository, and CVS will not create additional files in that directory. This condition can usually be cleared by using `update -d`.

'CVS/Notify'

Stores notifications that have not yet been sent to the server.

'CVS/Notify.tmp'

Temp file for '`Notify`'. The usual procedure for modifying '`Notify`' is to write out '`Notify.tmp`' and then rename it to '`Notify`'.

'CVS/Repository'

The path to the project-specific subdirectory in the repository. This may be an absolute path, or it may be relative to the path given in Root.

This file is always present.

'CVS/Root'

This is the repository; that is, the value of the $CVSROOT environment variable or the argument to the -d global option.

This file is always present.

'CVS/Tag'

If there is a sticky tag or date on this directory, it is recorded in the first line of the file. The first character is a single letter indicating the type of tag: T, N, or D, for branch tag, nonbranch tag, or date respectively. The rest of the line is the tag or date itself.

'CVS/Template'

Contains a log message template as specified by the rcsinfo file. (See ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ earlier in this chapter.) It is relevant only for remote working copies; working copies on the same machine as the repository just read rcsinfo directly.

'CVS/Update.prog'

Records the name of the program specified by the -u option in the modules file. (See the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter.)

Environment Variables

These are all the environment variables that affect CVS.

$COMSPEC

This is used in OS/2 only; it specifies the name of the command interpreter. It defaults to CMD.EXE.

$CVS_CLIENT_LOG

Used for debugging the client/server protocol. Set this variable to a file name before you start using CVS; all traffic to the server will be logged in filename.in, and everything from the server will be logged in filename.out.

$CVS_CLIENT_PORT

Used in Kerberos-authenticated client/server access.

$CVSEDITOR

Specifies the program to use to edit log messages for commits. This overrides `$EDITOR` and `$VISUAL`.

$CVSIGNORE

A whitespace-separated list of file names and wildcard patterns that CVS should ignore. (See also the -I option to the ⟨undefined⟩ [import], page ⟨undefined⟩ command.)

This variable is appended last to the ignore list during a command. The list is built up in this order: '`CVSROOT/cvsignore`', the '`.cvsignore`' file in your home directory, the `$CVSIGNORE` variable, any -I command option, and finally the contents of '`.cvsignore`' files in the working copy used as CVS works in each directory. A `!` as the ignore specification at any point nullifies the entire ignore list built up to that point.

$CVS_IGNORE_REMOTE_ROOT

Recently obsolete.

$CVS_PASSFILE

Tells CVS to use some file other than .cvspass in your home directory. (See the file '`.cvspass`' in the ⟨undefined⟩ [Ficheros de Control de Ejecucion], page ⟨undefined⟩ section in this chapter.)

$CVS_RCMD_PORT

Specifies the port number to contact the rcmd daemon on the server side. (This variable is currently ignored in Unix CVS clients.)

$CVSREAD

Makes working copy files read-only on checkout and update, if possible (the default is for them to be read-write). (See also the -r global option.)

$CVSROOT

This specifies the path to the repository. This is overridden with the -d global option and by the ambient repository for a given working copy. The path to the repository may be preceded by an access method, username, and host, according to the following syntax:

```
[[:METHOD:][[USER@]HOST]:]/REPOSITORY_PATH
```

See the -d global option, in the section ⟨undefined⟩ [Opciones Globales], page ⟨undefined⟩ near the beginning of this chapter, for a list of valid methods.

$CVS_RSH

Specifies an external program for connecting to the server when using the `:ext:` access method. Defaults to `rsh`, but `ssh` is a common replacement value.

$CVS_SERVER

Program to invoke for CVS on the server side. Defaults to `cvs`, of course.

$CVS_SERVER_SLEEP

Delays the start of the server child process by the specified number of seconds. This is used only for debugging, to allow time for a debugger to connect.

$CVSUMASK

Permissions for files and directories in the repository. (You probably don't want to set this; it doesn't work for client/server anyway.)

$CVSWRAPPERS

A whitespace-separated list of file names, wildcards, and arguments that CVS should use as wrappers. (See ⟨undefined⟩ [cvswrappers], page ⟨undefined⟩ in the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter for more information.)

$EDITOR

(See ⟨undefined⟩ [$CVSEDITOR], page ⟨undefined⟩.)

$HOME %HOMEDRIVE% %HOMEPATH%

Where the '`.cvsrc`', '`.cvspass`', and other such files are found (under Unix, only `$HOME` is used). In Windows NT, `%HOMEDRIVE%` and `%HOMEPATH%` might be set for you; in Windows 95, you may need to set them for yourself.

In Windows 95, you may also need to set `%HOME%`. Make sure not to give it a trailing backslash; use set `HOME=C:` or something similar.

$PATH

Obsolete.

$TEMP $TMP $TMPDIR

Where temporary files go (the server uses TMPDIR; Windows NT uses TMP). Setting this on the client side will not affect the server. Setting this on either side will not affect where CVS stores temporary lock files. (See ⟨undefined⟩ [config], page ⟨undefined⟩ in the ⟨undefined⟩ [Ficheros de Administracion del Repositorio], page ⟨undefined⟩ section in this chapter for more information.)

$VISUAL

(See ⟨undefined⟩ [$CVSEDITOR], page ⟨undefined⟩.)

Herramientas de terceros

Mucha gente ha escrito programas para mejorar CVS. Yo los llamo *herramientas de terceros* porque tienen sus propios mantenedores, independientes del equipo de desarrollo de CVS. La mayoría de estos programas no se distribuyen con CVS aunque algunos sí lo hacen. Este capítulo está dedicado a las herramientas de terceros que he encontrado útiles pero que no se distribuyen con CVS.

Aunque hay algunas interfaces para CVS no Unix en línea de órdenes que gozan de gran popularidad y son de amplio uso (los sitios de descarga aparecen listados en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩) en este capítulo no hablaremos de ellas. Su popularidad hace que resulte sencillo obtener información sobre ellas en listas de correo y grupos de noticias. Como clara excepción tenemos la interfaz pcl-cvs para Emacs que aunque resulta muy útil a veces es algo compleja de instalar.

pcl-cvs – Una interfaz de Emacs para CVS

Depende de: Emacs, Elib

URLs:

`ftp://rum.cs.yale.edu/pub/monnier/pcl-cvs/`

`ftp://ftp.lysator.liu.se/pub/emacs/pcl-cvs-1.05.tar.gz`

`ftp://ftp.red-bean.com/pub/kfogel/pcl-cvs-1.05.tar.gz`

Autores: Per Cederqvist y Stefan Monnier (mantenedor actual)

`pcl-cvs` es una de las dos interfaces de CVS para Emacs. La otra es la interfaz nativa VC (Control de Versiones) integrada en Emacs. Yo prefiero pcl-cvs porque se escribió para CVS en exclusiva y, por tanto, funciona muy bien con la manera que tiene CVS de hacer las cosas. VC, por otra parte, se diseñó para funcionar con sistemas de control de versiones muy diferentes (RCS y SCCS, así como CVS) y no está realmente "optimizado" para CVS. Por ejemplo, VC presenta una interfaz basada más en archivos que en directorios para el control de revisiones.

Las ventajas de pcl-cvs son suficientes como para que mucha gente elija descargarlo e instalarlo en vez de usar VC. Desgraciadamente pcl-cvs tiene dos desventajas: su instalación puede resultar algo compleja (una gran parte de esta sección está dedicada a solucionar posibles problemas durante la misma) y sus versiones más recientes se han mostrado algo inestables.

Este último problema es temporal pero hace que nos preguntemos qué versión usar. Stefan Monnier se ha hecho cargo recientemente del mantenimiento de pcl-cvs. La última versión, 2.9.6 (disponible en la primera URL de la lista anterior), resultaba un tanto problemática cuando la probé. No dude que los problemas desaparecerán pronto pero, mientras tanto, quizá quiera usar una versión más antigua. Aquí voy a documentar la versión 1.05 pues es la que he estado usando a diario durante bastante tiempo y se ha portado bastante bien. Por suerte los procesos de instalación no cambian mucho de versión en versión. Si decide utilizar pcl-cvs le sugiero que busque en el sitio de descarga de Monnier una versión más reciente que la 2.9.6; si la hay, pruébela antes de intentarlo con la 1.05.

Se habrá dado cuenta de que he puesto dos URLs para la versión 1.05. La primera es del sitio de Per Cederqvist, donde aún puede encontrarse un archivo antiguo de pcl-cvs.

No obstante, como no estoy seguro de por cuánto tiempo permanecerá su archivo en línea, también he puesto a disposición pública la versión 1.05 en ftp.red-bean.com.

Aunque en el resto de estas instrucciones se utilizan ejemplos tomados de la versión 1.05 deberían poder aplicarse igualmente a versiones posteriores.

Instalar pcl-cvs

Si no está acostumbrado a trastear con la instalación de Emacs ni con asuntos relacionados con el mantenimiento de sitios el proceso de instalación de pcl-cvs puede parecerle un tanto intimidatorio. Quizá le ayude algo de información acerca de cómo funciona Emacs.

La mayoría de las prestaciones de alto nivel de Emacs están escritas en un lenguaje llamado "Emacs Lisp" (Emacs en sí es esencialmente un intérprete de este lenguaje). La gente añade nuevas funcionalidades a Emacs distribuyendo archivos de código en Emacs Lisp. `pcl-cvs` está escrito en este lenguaje y depende de una librería de útiles funciones Emacs Lisp genéricas que se conoce como *Elib* (también escrita en parte por Per Cederqvist, pero distribuida de manera independiente a pcl-cvs).

Elib no se incluye en la distribución estándar de Emacs (al menos no en la del Emacs de la FSF; desconozco si ocurre lo mismo con XEmacs), por lo que tendrá que descargarla e instalarla usted mismo antes de que pueda utilizar pcl-cvs. Puede hacerse con ella en `ftp://ftp.lysator.liu.se/pub/emacs/elib-1.0.tar.gz`. Las instrucciones de instalación se encuentran en el mismo paquete.

Una vez se ha instalado Elib ya estamos listos para compilar e instalar pcl-cvs. Estas instrucciones son aplicables tanto a la versión 1.05 como a la serie 2.x (aunque debería comprobar los archivos NEWS e INSTALL de las distribuciones más recientes para ver qué ha cambiado).

Pare empezar desempaquete pcl-cvs (yo estoy usando la versión 1.05 pero podría tratarse perfectamente de la 2.9.6)

```
floss$ zcat pcl-cvs-1.05.tar.gz | tar xvf -
pcl-cvs-1.05/
pcl-cvs-1.05/README
pcl-cvs-1.05/NEWS
pcl-cvs-1.05/INSTALL
pcl-cvs-1.05/ChangeLog
pcl-cvs-1.05/pcl-cvs.el
pcl-cvs-1.05/pcl-cvs.texinfo
pcl-cvs-1.05/compile-all.el
pcl-cvs-1.05/pcl-cvs-lucid.el
pcl-cvs-1.05/pcl-cvs-startup.el
pcl-cvs-1.05/pcl-cvs.info
pcl-cvs-1.05/Makefile
pcl-cvs-1.05/texinfo.tex
```

y sitúese en el nivel más alto del directorio que contiene las fuentes:

```
floss$ cd pcl-cvs-1.05/
```

Aquí se le proporciona un Makefile. De acuerdo con las instrucciones del archivo INSTALL tiene que editar unas cuantas rutas en la parte superior del Makefile y ejecutar luego:

```
floss$ make install
```

Si eso funciona, fantástico. No obstante, esto a veces puede resultar en un error (aunque el propio código de pcl-cvs es muy portable los procedimientos para su instalación a veces no lo son tanto). De encontrarse con un error haga esto:

```
floss$ make clean
floss$ make
```

Si todo va bien estas órdenes llevan a cabo una parte significativa de la instalación compilando a "byte-code" todos los archivos Emacs Lisp. (Al compilar a "byte-code" convierte un archivo de código Emacs Lisp perfectamente legible, un archivo .el, en una representación más compacta y eficiente, un archivo .elc. Emacs puede cargar y ejecutar los archivos .elc con un mejor rendimiento que los archivos .el).

Continuaré como si la compilación a "byte-code" se hubiese llevado a cabo con éxito. Si no ha sido así no se preocupe: los archivos .elc son un lujo, no una necesidad. Mejoran ligeramente el rendimiento pero puede correr pcl-cvs directamente desde los archivos .el sin problemas.

Si el "make install" falló el siguiente paso es colocar el Emacs Lisp (.el o .elc) en un directorio donde Emacs pueda cargarlo automáticamente. Emacs tiene un directorio designado en el sistema para el Lisp instalado localmente. Para encontrar ese directorio (habrá un archivo llamado '`default.el`' en él) mire en los siguientes lugares por este orden:

1. /usr/share/emacs/site-lisp/
2. /usr/local/share/emacs/site-lisp/
3. /usr/lib/emacs/site-lisp/
4. /usr/local/lib/emacs/site-lisp/

En cuanto haya encontrado su directorio site-lisp copie todos los archivos Lips en él (puede necesitar ser root para hacer esto):

```
floss# cp -f *.el *.elc /usr/share/emacs/site-lisp/
```

El último paso es comunicarle a Emacs los puntos de entrada a pcl-cvs (siendo el principal la función cvs-update) para que sea capaz de cargar el código de pcl-cvs bajo demanda. Como Emacs siempre lee el archivo default.el cuando arranca ahí es donde necesitará listar los puntos de entrada a pcl-cvs.

Por suerte pcl-cvs ya viene con el contenido necesario para default.el. Simplemente coloque el contenido de pcl-cvs-startup.el en default.el (o quizá en su .emacs si sólo lo instala para usted) y reinicie Emacs.

Quizá quiera copiar también los archivos .info a su árbol info y añadir pcl-cvs al índice de contenidos del archivo dir.

Usar pcl-cvs

Una vez instalado pcl-cvs es muy fácil de usar. Ejecute simplemente la función cvs-update y pcl-cvs le mostrará un búfer con los archivos de su copia de trabajo que se hayan modificado o actualizado. A partir de ahí puede hacer "commits", "diffs" y demás.

Al ser cvs-update el punto de entrada principal le sugiero que lo enlace con un atajo de teclado conveniente antes de que continuemos. Yo lo tengo enlazado a *`Ctrl+c v`* en mi .emacs:

```
(global-set-key "\C-cv" 'cvs-update)
```

De otra manera también puede ejecutarlo tecleando *M-x cvs-update* (también conocido como *Esc-x cvs-update*).

Cuando invoca a cvs-update éste ejecuta cvs update como si estuviese en el directorio del archivo que se encuentre en el búfer (tal y como si hubiese tecleado cvs update estando en ese directorio desde la línea de órdenes). He aquí un ejemplo de lo que podría ver desde Emacs:

```
PCL-CVS release 1.05 from CVS release $Name:  $.
Copyright (C) 1992, 1993 Per Cederqvist
Pcl-cvs comes with absolutely no warranty; for details consult the manual.
This is free software, and you are welcome to redistribute it under certain
conditions; again, consult the TeXinfo manual for details.
 Modified ci README.txt
 Modified ci fish.c
---------- End ----
```

Se han modificado dos archivos localmente (en algunas versiones de pcl-cvs se muestran los subdirectorios donde se encuentran los archivos). Lo siguiente es hacer "commit" sobre ambos archivos o sobre uno de ellos, que es lo que significa el ci en cada línea. Vaya a su línea y escriba *c*. Se le conducirá a un búfer de mensajes de cambios donde podrá escribir un mensaje de cambios tan largo como quiera (la edición real de los mensajes con los cambios es la mayor ventaja de pcl-cvs sobre la línea de órdenes). Introduzca *Ctrl+c Ctrl+c* cuando haya acabado para completa el "commit".

Si desea hacer "commit" sobre múltiples archivos a la vez de manera que compartan un mismo mensaje de cambios use primero m para marcar los archivos sobre los que pretendar hacer un "commit". Aparecerá un asterisco junto a cada archivo conforme los vaya marcando:

```
PCL-CVS release 1.05 from CVS release $Name:  $.
Copyright (C) 1992, 1993 Per Cederqvist
Pcl-cvs comes with absolutely no warranty; for details consult the manual.
This is free software, and you are welcome to redistribute it under certain
conditions; again, consult the TeXinfo manual for details.
* Modified ci README.txt
* Modified ci fish.c
---------- End ----
```

Ahora cuando escriba c en cualquier lugar se aplicará a todos los archivos marcados (y únicamente a ellos). Escriba el mensaje con los cambios y haga "commit" con *Ctrl+C Ctrl+C* como antes.

También puede escribir *d* para ejecutar cvs diff sobre un archivo (o sobre unos archivos marcados) y *f* para editarlo con Emacs. Hay más órdenes disponibles, use *Ctrl+h* m en el búfer de actualización para ver qué más puede hacer.

Manejo de errores en pcl-cvs

El programa pcl-cvs ha tenido históricamente una manera un tanto singular de manejar los mensajes de error e informativos de CVS (aunque quizá esto se haya corregido en las últimas versiones). Cuando se encuentra con un mensaje de CVS que no entiende se pone

histérico y le envía a un búfer de correo listo para enviar un informe de error previamente generado al autor de pcl-cvs. Desgraciadamente, entre los mensajes de CVS que pcl-cvs puede no entender se encuentran los asociados con la resolución de conflictos los cuales, aunque no son muy comunes, ocurren ciertamente de vez en cuando.

Si pcl-cvs le envía repentinamente a un búfer de correo no se asuste. Lea el contenido del búfer con cuidado (la salida de CVS culpable debería aparecer en algún lugar. Si parece una fusión puede deshacerse tranquilamente del búfer y volver a ejecutar cvs-update. Ahora debería tener éxito al no tener que mostrar ya CVS ningún mensaje sobre la fusión (porque ya ha tenido lugar).

(Actualización: este problema parece haberse arreglado en las versiones más recientes de pcl-cvs por lo que puede ignorar este aviso casi con total seguridad).

El futuro pcl-cvs

Aunque pueda estar dándole la impresión de que pcl-cvs apenas se mantiene y supone una inversión de riesgo la inestabilidad parece ser temporal. Stefan Monnier es un mantenedor despierto (me he puesto en contacto con él muchas veces mientras escribía este capítulo y siempre me ha respondido; ya se encuentra en vías de corregir algunos errores de la versión 2.9.6). Seguramente cuando esto se publique podrá descargar la versión 2.9.7 o posterior con toda tranquilidad.

De hecho, recibí hace poco un correo de Greg Woods, un mantenedor anterior de pcl-cvs, sobre este asunto que reproduzco aquí con su permiso:

```
De: woods@most.weird.com (Greg A. Woods)
Sobre: Re:  Estado del mantenimiento de pcl-cvs, estabilidad de las
"versiones" recientes ?
Para: kfogel@red-bean.com
Fecha: Sáb, 29 Ago 1999 18:59:19 -0400 (EDT)

[...]
He estado usando las versiones de Stefan desde hace ya tiempo y de hecho
he abandonado mi propia rama.

Ha hecho un trabajo realmente bueno con PCL-CVS y excepto por algunos
extraños errores en la versión 2.9.6 lo uso a diario y me resulta
bastante usable (e infinitamente más usable con el CVS moderno que con
el que venía con la distribución de CVS! ;-).

He añadido un archivo pcl-cvs.README a mi sitio FTP para indicar que los
archivos que hay son bastante antiguos (al menos según el tiempo de
Internet! ;-) y también para proporcionar un enlace al sitio FTP de
Stefan.

[...]
```

En un mensaje posterior Greg me dijo que la FSF está considerando incluir pcl-cvs en su próxima versión de Emacs (20.5) lo que dejaría obsoletos los consejos anteriores sobre su instalación. En fin, a veces es difícil lidiar con el software libre.

cvsutils – Utilidades genéricas para usar con CVS

Dependen de: Perl

URLs:

`http://www.red-bean.com/cvsutils`

Autores: Tom Tromey (autor original) y Pavel Roskin (mantenedor actual)

El conjunto de pequeños programas conocido como `cvsutils` genralmente (aunque no siempre) llevan a cabo operaciones *estando desconectados* en la copia de trabajo de CVS. Las operaciones estando desconectado son aquellas que pueden realizarse sin tener que ponerse en contacto con el repositorio al tiempo que se mantiene la copia de trabajo en un estado consistente para la próxima vez que contactemos con el repositorio. El comportamiento sin línea puede llegar a sernos extremadamente útil cuando nuestra conexión de red con el repositorio sea lenta o nada eficaz.

Los programas cvsutils se listan abajo en un orden de utilidad aproximado (según mi opinión) siendo los primeros los más útiles. Esto también los ordena según su seguridad. La seguridad es algo a tener muy en cuenta ya que algunas de estas utilidades pueden causar pérdidas de modificaciones en local o archivos de su copia de trabajo aún operando de manera correcta. Por esto, lea las descripciones cuidadosamente antes de usar estas utilidades.

Esta documentación se centra en la versión 0.1.4. Asegúrese de leer el archivo README en cualquier versión posterior para una información más actualizada.

cvsu

Nivel de peligrosidad: ninguna

Contacta con el repositorio: no

Lleva a cabo un cvs update estando desconectado comparando las marcas temporales de los archivos del disco con las registradas en CVS/Entries. Así puede averiguar qué archivos se han modificado localmente y cuáles parece no controlar CVS. A diferencia de `cvs update` cvsu no se trae los cambios del repositorio.

Aunque acepta varias opciones cvsu suele invocarse la mayoría de las veces sin ellas:

```
floss$ cvsu
? ./bar
? ./chapter-10.html
M ./chapter-10.sgml
D ./out
? ./safe.sh
D ./tools
```

Los códigos de la izquierda son como la salida de cvs update excepto `D` que signfica directorio. En este ejemplo se muestra que chapter-10.sgml se ha modificado localmente. Lo que no aparece en el ejemplo es que cvsu se ejecutó instantáneamente mientras que un cvs update normal hubiese requerido medio minuto más o menos con una conexión lenta. Ejecute

```
floss$ cvsu --help
```

para ver una lista de opciones.

cvsdo

Nivel de peligrosidad: casi ninguna

Contacta con el repositorio: no

Esto puede simular los efectos de cvs add y cvs remove sobre nuestra copia de trabajo pero sin ponerse en contacto con el repositorio. Está claro que aún tendrá que hacer "commit" de los cambios para que tengan efecto en el repositorio pero al menos las órdenes add y remove en sí mismas pueden acelerarse de esta manera. He aquí como usarlo

```
floss$ cvsdo add FILENAME
```

o

```
floss$ cvsdo remove FILENAME
```

Para ver una lista con el resto de opciones ejecute:

```
floss$ cvsdo --help
```

cvschroot

Nivel de peligrosidad: baja

Contacta con el repositorio: no

Esto maneja un movimiento del repositorio haciendo que nuestra copia de trabajo apunte al nuevo. Esto resulta útil cuando se copia un repositorio en masa a un nuevo lugar. Cuando eso sucede no suele afectar a ninguna de las revisiones pero los archivos CVS/Root (y posiblemente CVS/Repository) de cada copia de trabajo deben actualizarse para que apunten al nuevo emplazamiento. Usar cvschroot es mucho más rápido que traerse una nueva copia. Otra ventaja es que no perderá los cambios que pueda haber hecho en local.

Uso:

```
floss$ cvschroot NEW_REPOS
```

Por ejemplo:

```
floss$ cvschroot :pserver:newuser@newhost.wherever.com:/home/cvs/myproj
```

cvsrmadm

Nivel de peligrosidad: bajo a medio

Contacta con el repositorio: no

Esto elimina todos los subdirectorios administrativos CVS/ de su copia de trabajo dejando una estructura similar a la que crea cvs export.

Aunque no perderá ningún cambio local usando cvsrmadm su copia de trabajo dejará de serlo.

Úselo con precaución.

cvspurge

Nivel de peligrosidad: medio

Contacta con el repositorio: no

Elimina todos los archivos no controlados por CVS de su copia de trabajo. No deshace ningún cambio local en los archivos que controle CVS.

Uselo con precaución.

cvsdiscard

Nivel de peligrosidad: medio a alto

Contacta con el repositorio: quizá

Complemento de cvspurge, en vez de eliminar los archivos desconocidos pero manteniendo los cambios locales cvsdiscard deshace los cambios hechos en local (sustituyendo esos archivos con copias nuevas del repositorio) pero manteniendo los archivos desconocidos.

Uselo con extrema precaución.

cvsco

Nivel de peligrosidad: alto

Contacta con el repositorio: quizá

Esto es la unión de cvspurge y cvsdiscard. Deshace cualquier cambio local y elimina los archivos desconocidos de la copia de trabajo.

Uselo con precaución auténticamente paranoica.

cvsdate

Este guión se encuentra aparentemente incompleto y quizá nunca se acabe. (Acuda al archivo README para más detalles).

cvs2cl – Genera ChangeLogs al estilo GNU

Depende de: Perl

URL: `http://www.red-bean.com/~kfogel/cvs2cl.shtml`

cvs2cl.pl condensa y reformatea la salida del registro de cvs para crear un archivo ChangeLog al estilo GNU para su proyecto. Los ChangeLogs son documentos organizados cronológicamente en los que se muestra el historial de cambios de un proyecto con un formato diseñado especialmente para su legibilidad (fíjese en los siguientes ejemplos).

El problema con la orden `cvs log` es que presenta su salida basándose en cada archivo sin tener en cuenta que un mismo mensaje de cambios, si aparece prácticamente al mismo tiempo en archivos diferentes, implica que esas revisiones formaron parte de un único "commit". Por esto, resulta desesperante ir leyendo la salida del registro para hacerse una idea del estado en el que se encuentra el desarrollo del proyecto. Realmente sólo puede ver la historia de un archivo al mismo tiempo.

En el ChangeLog producido por cvs2cl.pl los mensajes de cambios idénticos se unen de manera que si un único "commit" implica a un grupo de archivos eso aparece como una única entrada. Por ejemplo:

```
floss$ cvs2cl.pl -r
cvs log: Logging .
cvs log: Logging a-subdir
cvs log: Logging a-subdir/subsubdir
cvs log: Logging b-subdir
```

```
floss$ cat ChangeLog
...
1999-08-29 05:44  jrandom

   * README (1.6), hello.c (2.1), a-subdir/whatever.c (2.1),
   a-subdir/subsubdir/fish.c (2.1): Haciendo commit desde pcl-cvs 2.9,
sólo para ir abriendo boca.

1999-08-23 22:48  jrandom

   * README (1.5): [no log message]

1999-08-22 19:34  jrandom

   * README (1.4): trivial change
...
floss$
```

La primera entrada muestra que se hizo "commit" de cuatro archivos al mismo tiempo con el mensaje de cambios "Haciendo commit desde pcl-cvs 2.9 sólo para ir abriendo boca". (Se usó la opción -r para mostrar el número de revisión de cada archivo asociado a ese mensaje de cambios.)

Al igual que el propio CVS, cvs2cl.pl toma el directorio actual como una variable implícita pero trabaja sobre archivos individuales si se le proporcionan variables de nombre de archivo. Éstas son algunas de las opciones más usadas:

`h`, `--help`

Muestra su uso (incluyendo una completa lista de opciones).

`-r`, `--revisions`

Muestra los números de revisión en la salida. Si se usa junto con -b las ramas se muestran como NOMBREDELARAMA.N siendo N la revisión de esa rama.

`-t`, `--tags`

Muestra las etiquetas (nombres simbólicos) en las revisiones que las tengan.

`-b`, `--branches`

Muestra el nombre de la rama para las revisiones de esa rama. (Vea también -r.)

`-g OPTS`, `--global-opts OPTS`

Pasa OPTS como una variable global para cvs. Internamente cvs2cl.pl invoca a cvs para hacerse con los datos de registro en crudo; es entonces cuando OPTS se pasa al cvs en esa invocación. Por ejemplo, para obtener un comportamiento tranquilo y compresión, puede hacer esto:

```
floss$ cvs2cl.pl -g "-Q -z3"
```

`-l OPTS`, `--log-opts OPTS`

Similar a -g, sólo que OPTS se pasa como opciones de la orden en vez de como opciones globales. Para generar un ChangeLog en el que aparezcan únicamente los "commits" que tuvieron lugar entre el 26 de Julio y el 15 de Agosto puede hacer esto:

```
floss$ cvs2cl.pl -l "'-d1999-07-26<1999-08-15'"
```

Fíjese en el entrecomillado doble, necesario en Unix porque la shell que invoca a cvs log (desde dentro de cvs2cl.pl) interpreta el < como un símbolo de redirección en shell. Por esto, las comillas tienen que colocarse como parte de la variable haciéndose necesario envolverlo todo con unas comillas adicionales.

`-d`, `--distributed`

Coloca un ChangeLog individual en cada subdirectorio cubriendo sólo los "commits" a ese subdirectorio (contrario a crear un ChangeLog que cubra el directorio desde el que se invoca cvs2cl.pl y todos sus subdirectorios).

cvsq – Encola órdenes CVS para una posterior conexión

Depende de: Bash

URL: `http://www.volny.cz/v.slavik/lt/cvsq.html`

Esto es lo que tiene que decir Vaclav Slavik <v.slavik@volny.cz>, el autor de cvsq, sobre él:

cvsq significa "cvs encolado" y es un pequeño guión de bash que envuelve el CVS de Cyclic. Hace más fácil el trabajo con repositorios CVS para la gente con conexiones de marcado telefónico porque permite encolar órdenes CVS y pasárselas al "verdadero cvs" posteriormente.

Por ejemplo, puede hacer "commit" sobre los archivos inmediatamente tras editarlos, estando desconectado, así que no se olvide de ellos:

```
cvsq commit -m "change 1" file1.c
cvsq commit -m "change 2" file2.c
cvsq commit -m "change 3" file3.c
```

Y luego, cuando se conecte, sólo tendrá que escribir:

```
cvsq upload
```

y todos los cambios se aplicarán en el repositorio. Si ocurre algún error durante el envío de algún archivo éste no se perderá. En vez de eso aparecerá un mensaje de error y el archivo continuará en la cola de cvsq.

Puede utilizar cvsq incluso con órdenes que no tengan sentido estando desconectado. En ese caso, la orden se pasa inmediatamente a cvs y no se encola. Por ejemplo, puede hacer un cvsq update y éste no se pondrá en la cola sino que se ejecutará inmediatamente. De hecho, puede comenzar a utilizar cvsq como un sustituto de cvs.

cvsq es de dominio público.

cvslock – Bloquea los repositorios para evitar la atomicidad

Depende de: compilador de C para la instalación, nada para la ejecución

URL: `ftp://riemann.iam.uni-bonn.de/pub/users/roessler/cvslock/`

Este programa bloquea un repositorio CVS (ya sea su lectura o la escritura en él) de la misma manera en que lo hace CVS por lo que éste respetará el bloqueo. Esto puede resultar útil cuando, por ejemplo, necesita realizar una copia del repositorio completo y quiera evitar capturar partes de "commits" o archivos de bloqueo de otra gente.

La distribución de cvslock se encuentra excelentemente empaquete y puede instalarse de acuerdo con los procedimientos GNU habituales. He aquí la transcripción de una sesión de instalación típica:

```
floss$ zcat cvslock-0.1.tar.gz | tar xvf -
cvslock-0.1/
cvslock-0.1/Makefile.in
cvslock-0.1/README
cvslock-0.1/COPYING
cvslock-0.1/Makefile.am
cvslock-0.1/acconfig.h
cvslock-0.1/aclocal.m4
cvslock-0.1/config.h.in
cvslock-0.1/configure
cvslock-0.1/configure.in
cvslock-0.1/install-sh
cvslock-0.1/missing
cvslock-0.1/mkinstalldirs
cvslock-0.1/stamp-h.in
cvslock-0.1/cvslock.c
cvslock-0.1/cvslock.1
cvslock-0.1/snprintf.c
cvslock-0.1/cvslssh
cvslock-0.1/VERSION
floss$ cd cvslock-0.1
floss$ ./configure
 ...
floss$ make
gcc -DHAVE_CONFIG_H -I. -I. -I.   -g -O2 -c cvslock.c
gcc -g -O2  -o cvslock  cvslock.o
floss$ make install
 ...
floss$
```

(Tenga en cuenta que quizá necesite llevar a cabo el make install como root).

Ahora cvslock ya se encuentra instalado como /usr/local/bin/cvslock. Cuando lo invoque puede especificar el repositorio con -d o mediante la variable de entorno $CVSROOT tal y como haría con CVS (en los siguientes ejemplo se usa -d). La única variable que se requiere es el nombre de directorio a bloquear en relación con la raíz del repositorio. En este ejemplo no hay subdirectorios por lo que sólo se crea un archivo de bloqueo:

```
floss$ ls /usr/local/newrepos/myproj/b-subdir/
```

```
random.c,v
floss$ cvslock -d /usr/local/newrepos  myproj/b-subdir
floss$ ls /usr/local/newrepos/myproj/b-subdir/
#cvs.rfl.cvslock.floss.27378  random.c,v
floss$ cvslock -u -p 27378 -d /usr/local/newrepos  myproj/b-subdir
floss$ ls /usr/local/newrepos/myproj/b-subdir/
random.c,v
floss$
```

Fíjese en que cuando retiré el bloqueo (-u para `desbloquear`) tuve que especificar `-p 27378`. Eso es porque cvslock usa los ID de los procesos de Unix al crear los nombres de los archivos de bloqueo para asegurarse de que sus bloqueos son únicos. Al desbloquear tiene que counicarle a cvslock qué instancia de bloqueo eliminar incluso si sólo hay una. Por esto, la opción -p le dice a cvslock de qué instancia previa de sí mismo se está deshaciendo (no obstante, puede usar -p con o sin -u).

Si va a estar trabajando con el repositorio por un tiempo, realizando varias operaciones directamente en el sistema de archivos, puede usar la opción -s para que cvslock inicie una nueva shell por usted. Consultará entonces la variable de entorno `$SHELL` en su shell actual para determinar qué intérprete de órdenes usar:

```
floss$ cvslock -s -d /usr/local/newrepos myproj
```

Los bloqueos permanecen hasta que salga del intérprete, momento en el que se eliminarán automáticamente. También puede utilizar la opción -c para ejecutar orden mientras se bloquea el repositorio. Al igual que con -s, los archivos de bloqueo se colocan antes de que se inicie la orden y se retiran una vez haya acabado. En el siguiente ejemplo bloqueamos el repositorio durante el tiempo suficiente para mostrar una lista con todos los archivos de bloqueo:

```
floss$ cvslock -c 'find . -name "*cvslock*" ' -d /usr/local/newrepos myproj
cvslock: '/usr/local/newrepos/myproj' locked successfully.
cvslock: Starting 'find . -name "*cvslock*" -print'...
./a-subdir/subsubdir/#cvs.rfl.cvslock.floss.27452
./a-subdir/#cvs.rfl.cvslock.floss.27452
./b-subdir/#cvs.rfl.cvslock.floss.27452
./#cvs.rfl.cvslock.floss.27452
floss$ find /usr/local/newrepos/myproj -name "*cvslock*" -print
floss$
```

La orden (el parámetro de la opción -c) se ejecuta con el directorio del repositorio especificado como su directorio de trabajo.

De manera predeterminada cvslock crea bloqueos de lectura. Puede decirle que use bloqueos de escritura mediante la opción -W. (Puede pasarle -R para especificar bloqueos de lectura pero de todas formas ése es el comportamiento predeterminado.) Quite siempre todos los bloqueos una vez haya acabado de manera que los procesos CVS del resto de usuarios no tengan que esperar innecesariamente.

Tenga en cuenta que cvslock debe ejecutarse en la máquina en la que reside el repositorio, no puede especificar un repositorio remoto. (Para más información ejecute `man cvslock`, página de manual que se habrá instalado al hacer `make install`.)

Otros paquetes

Hay disponibles muchos otros paquetes de terceros para CVS. Los siguientes son algunos de ellos.

CVSUp (Parte del proyecto FreeBSD)

CVSUp es una eficiente herramienta de replicado genérico con soporte especial integrado para replicar repositorios CVS. El sistema operativo FreeBSD lo usa para distribuir los cambios desde su repositorio principal de manera que los usuarios pueden mantenerse actualizados de una manera conveniente.

Para más información sobre CVSUp en general acuda a `http://www.polstra.com/projects/freeware/CVS`

Para su uso en FreeBSD en particular mire `http://www.freebsd.org/handbook/synching.html#CVSUP`.

CVSWeb: una interfaz web para repositorios CVS

CVSWeb ofrece una interfaz web para moverse por repositorios CVS. Un nombre más adecuado puede ser "RCSWeb" porque lo que realmente hace es permitirle es moverse por las revisiones directamente en un repositorio viendo los mensajes de cambios y los diffs. Aunque nunca he encontrado que sea una interfaz particularmente convincente tengo que admitir que es suficientemente intuitiva y se usa en muchos sitios.

Aunque el software lo desarrolló originalmente Bill Fenner, la versión que se encuentra actualmente en un desarrollo más activo es la de Henner Zeller en `http://linux.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/`.

Quizá también quiera visitar el sitio original de Fenner `http://www.freebsd.org/~fenner/cvsweb/` y posiblemente el resúmen de lo que se cuece entorno a CVSWeb que realiza Cyclic Software en `http://www.cyclic.com/cyclic-pages/web-cvsweb.html`.

Finalmente, si le apeteciese ver a CVSWeb en acción, puede encontrar un buen ejemplo en `http://sourceware.cygnus.com/cgi-bin/cvsweb.cgi/`.

El directorio contrib/ de CVS

Tal y como mencionamos en ⟨undefined⟩ [Administracion del Repositorio], page ⟨undefined⟩ CVS viene con unas cuantas herramientas de terceros que se recogen en el directorio contrib/. Aunque no conozco ninguna regla formal para determinar qué herramientas se distribuyen con CVS puede estar en marcha un esfuerzo para recopilar las herramientas de terceros más usadas y colocarlas en contrib/ para que la gente sepa dónde encontrarlas. Hasta que eso suceda la mejor manera de encontrar esas herramientas es mirar en contrib/, en varios sitios web sobre CVS y preguntar en la lista de correo.

Escribir sus propias herramientas

CVS puede parecer a veces una desconcertante recopilación de estándares improvisados. Tenemos el formato RCS, varios formatos de salida (histórico, anotado, de registro, de actualización y muchos otros), muchos formatos de archivo para la administración de repositorios, formatos de archivo para las copias de trabajo, el protocolo cliente/servidor,

el protocolo de los archivos de bloqueo... (Se ha mareado ya? Pues sabe bien qué podría seguir y seguir.)

Por suerte estos estándares guardan bastante consistencia de versión en versión por lo que si quiere escribir una herramienta para trabajar con CVS al menos no tendrá que preocuparse de ir detrás de un objetivo en constante movimiento. Para cada estándar interno suelen haber unas pocas personas en la lista de correo `info-cvs@gnu.org` que lo conocen extremadamente bien (muchos de ellos me ayudaron durante la redacción de este libro). También hay documentación que viene con con la distribución de CVS (especialmente doc/cvs.texinfo, doc/cvsclient.texi y doc/RCSFILES). Finalmente está el código fuente de CVS en sí, la última palabra sobre cualquier cuestión relativa a la implementación o a su comportamiento.

Con todo esto a su disposición no hay razón para las dudas. Si se le ocurre alguna utilidad que haría la vida con CVS más fácil escríbala. Es muy probable que haya más gente que también la quiera. A diferencia de un cambio al propio CVS una pequeña utilidad externa puede alcanzar una mayor distribución con gran rapidez resultando en una retroalimentación más rápida para el autor así como una corrección más rápida de los errores para todos los usuarios.

Índice

Perdón, el índice está todavía en desarrollo.

Aunque el formato "online" permite búsquedas, decidí que la incompletud del índice no debía retrasar la publicación de los capítulos. Espero tener el índice terminado razonablemente pronto. También son bienvenidos los indexadores voluntarios – por favor, mande un email a `bug-cvsbook@red-bean.com` si está interesado.

Appendix A GNU General Public License

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors'

reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

* a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

* b) You must cause any work that you distribute or publish, that in

whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

* c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

* a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

* b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

* c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so

as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software

Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,

```
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when
it starts in an interactive mode:

Gnomovision version 69, Copyright (C) yyyy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, the commands you use may be
called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Library General Public
License instead of this License.
```

Appendix B GNU Free Documentation License

GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics,

a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you

distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
* D. Preserve all the copyright notices of the Document.
* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
* H. Include an unaltered copy of this License.
* I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
* K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
* M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
* N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as

invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this

License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be

similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

El siguiente texto traduce la licencia arriba indicada. Note, sin embargo que la traducción no tiene ningún valor legal, ni ha sido comprobada de acuerdo a la legislación de ningún país en particular. Se incluye sólo con propósitos educativos. Para efectos legales por favor remítase al original en inglés.

Version 1.1, Marzo 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Se permite la copia y distribución de copias literales de este documento de licencia, pero no se permiten cambios.

1. PREÁMBULO El propósito de esta licencia es hacer que un manual,

libro de texto, u otro documento escrito sea libre en el sentido de libertad: para asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, bien de manera comercial o no comercial. En segundo término, esta licencia preserva para el autor o para quien publica una manera de obtener reconocimiento por su trabajo, al tiempo que no es considerado responsable de las modificaciones realizadas por terceros. Esta licencia es una especie de "copyleft" que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Esta licencia complementa la Licencia Pública General GNU, que es una licencia de copyleft diseñada para el software libre. Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: un programa libre debe venir con los manuales que ofrezcan la mismas libertades que da el software. Pero esta licencia no se limita a manuales de software; puede ser usada para cualquier trabajo textual, sin tener en cuenta su temática o si se publica como libro impreso. Recomendamos esta licencia principalmente para trabajos cuyo propósito sea instructivo o de referencia.

2. APLICABILIDAD Y DEFINICIONES Esta Licencia se aplica a cualquier manual u otro trabajo que contenga una nota del propietario de los derechos de reproducción que indique que puede ser distribuido bajo los términos de esta Licencia. El "Documento", en adelante, se refiere a cualquiera de dichos manuales o trabajos. Cualquier miembro del público es un licenciatario, y será denominado como "Usted". Una "Versión Modificada" del Documento designa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma. Una "Sección Secundaria" es un apéndice titulado o una sección preliminar al prólogo del Documento que tiene que ver exclusivamente con la relación de quien publica o los autores del Documento con el tema general del Documento (o asuntos relacionados) y cuyo contenido no entra directamente en tal tema general. (Por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar matemáticas.) La relación puede ser un asunto de conexión histórica, o de posición legal, comercial, filosófica, ética o política con el tema o con materias relacionadas.

Las "Secciones Invariantes" son ciertas Secciones Secundarias cuyos títulos son denominados como Secciones Invariantes, en la nota que indica que el documento es liberado bajo esta Licencia.

Los "Textos de Cubierta" son ciertos pasajes cortos de texto que se listan, como Textos de Portada o Textos de Contra Portada, en la nota que indica que el documento es liberado bajo esta Licencia.

Una copia "Transparente" del Documento significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público general, cuyos contenidos pueden ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por pixeles) con programas genéricos para gráficas

o (para dibujos) algún editor de dibujos ampliamente disponible, y que sea adecuado para exportar a formateadores de texto o para traducción automática a una variedad de formatos adecuados para ingresar a formateadores de texto. Una copia hecha en un formato que de otra forma sería Transparente pero cuyo formato ha sido diseñado para impedir o dificultar subsecuentes modificaciones por parte de los lectores no es Transparente. Una copia que no es "Transparente" es llamada "Opaca".

Los ejemplos de formatos adecuados para copias Transparentes incluyen ASCII plano sin formato, formato de Texinfo, formato de LaTeX, SGML o XML que usen un DTD disponible ampliamente, y HTML simple que siga los estándares y esté diseñado para modificaciones humanas. Los formatos Opacos incluyen PostScript, PDF, formatos propietarios que pueden ser leídos y editados unicamente con procesadores de palabras propietarios, SGML o XML para los cuáles los DTD y/o herramientas de procesamiento no están disponibles generalmente, y el HTML generado en una máquina, producido por algún procesador de palabras solo con propósitos de presentación.

La "Portada" significa, para un libro impreso, la portada misma más las páginas siguientes necesarias para mantener, legiblemente, el material que esta Licencia requiere que aparezca en la portada. Para trabajos en formatos que no tienen Portada como tal, "Portada" significa el texto cerca a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del texto.

3. COPIA LITERAL Usted puede copiar y distribuir el Documento en cualquier medio, sea en forma comercial o no comercial, siempre y cuando esta Licencia, las notas de derecho de autor, y la nota de licencia que indica que esta Licencia se aplica al Documento se reproduzcan en todas las copias, y que usted no adicione ninguna otra condición sobre las expuestas en en esta Licencia. No puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3. Usted también puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

4. COPIADO EN CANTIDADES Si publica copias impresas del Documento que sobrepasen las 100, y la nota de Licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible, todos esos textos de Cubierta: Textos de Portada en la portada, y Textos de Contra Portada en la contra portada. Ambas cubiertas deben identificarlo a usted clara y legiblemente como quien publica tales copias. La portada debe presentar el título completo con todas las palabras del título igualmente prominentes y visibles. Usted puede adicionar otro material en las cubiertas. Las copias con cambios limitados a las cubiertas,

siempre que preserven el título del Documento y satisfagan estas condiciones, puede considerarse como copia literal. Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros listados (tantos como sea razonable colocar) en la cubierta real, y continuar con el resto en páginas adyacentes. Si publica o distribuye copias Opacas del Documento cuya cantidad excedaí las 100, debe incluir una copia Transparente que pueda ser leída por una máquina con cada copia Opaca, o indicar en o con cada copia Opaca una dirección en una red de computadores publicamente accesible que contenga una copia completa y Transparente del Documento, libre de material adicional, a la cual el público general de la red tenga acceso para bajar anónimamente sin cargo, usando protocolos de redes públicos y estándares. Si usted hace uso de la última opción, deberá tomar medidas razonablemente prudentes, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio indicado por lo menos un año después de su última distribución al público de copias Opacas de esa edición (directamente o a través de sus agentes o distribuidores). Se solicita, aunque no es requisito, que contacte a los autores del Documento antes de redistribuir cualquier gran número de copias, para permitirle la oportunidad de que le provean una versión actualizada del Documento.

5. MODIFICACIONES Usted puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las secciones 2 y 3 anteriores, siempre que usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada asumiendo el rol del Documento, por lo tanto licenciando la distribución y modificación de la Versión Modificada a quienquiera que posea una copia de este. En adición, debe hacer lo siguiente en la Versión Modificada:

1. Uso en la Portada (y en las cubiertas, si hay alguna) de un título distinto al del Documento, y de versiones anteriores (que deberían, si hay alguna, estar listados en la sección de Historia del Documento). Puede usar el mismo título que versiones anteriores del original siempre que quién publicó la primera versión lo permita.

2. Listar en la Portada, como autores, una o más personas o entidades responsables por la autoría o las modificaciones en la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (Todos sus autores principales, si hay menos de cinco).

3. Establecer en la Portada del nombre de quién publica la Versión Modificada, como quien publica.

4. Preservar todas las notas de derechos de reproducción del Documento.

5. Adyacente a las otras notas de derecho de reproducción, adicionar una nota de derecho de reproducción de acuerdo a sus modificaciones.

6. Incluir, immediatamente después de la nota de derecho de reproducción, una nota de licencia dando el permiso público para usar la Versión Modificada bajo los términos de esta Licencia, de la forma mostrada más adelante en el Addendum.

7. Preservar en esa nota de licencia el listado completo de Secciones Invariantes y de los Textos de las Cubiertas que sean requeridos como se especifique en la nota de Licencia del Documento.

8. Incluir una copia sin modificación de esta Licencia.

9. Preservar la sección con título "Historia", y su título, y adicionar a esta una sección estableciendo al menos el título, el año,los nuevos autores, y quién publicó la Versión Modificada como reza en la Portada. Si no hay una sección titulada "Historia" en el Documento, crear una estableciendo el título, el año, los autores y quien publicó el Documento como reza en la Portada, añadiendo además un artículo describiendo la Versión Modificada como se estableció en la oración anterior.

10. Preservar la localización en red, si hay , dada en el Documento para acceso público a una copia Transparente del Documento, así como las otras direcciones de red dadas en el Documento para versiones anteriores en las cuáles estuviese basado. Estas pueden ubicarse en la sección "Historia". Se puede omitir la ubicación en red para un trabajo publicado por lo menos 4 años antes que el Documento mismo, o si quien publicó originalmente la versión a la que se refiere da permiso.

11. En cualquier sección titulada "Agradecimientos" o "Dedicatorias", preservar el título de la sección, y preservar en la sección toda la sustancia y el tono de los agradecimientos y/o dedicatorias de cada contribuyente que estén incluídas.

12. Preservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Números de sección o el equivalente no son considerados parte de los títulos de la sección.

13. Borrar cualquier sección titulada "Aprobaciones". Una tal sección no pueden estar incluida en las Versiones Modificadas.

14. No retitular ninguna sección existente como "Aprobaciones" o conflictuar con título de alguna Sección Invariante.

Si la Versión Modificada incluye secciones o apendices nuevos o preliminares al prólogo que califican como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, adicione sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección. Puede adicionar una sección titulada "Aprobaciones", siempre que contenga unicamente aprobaciones de su Versión Modificada por varias fuentes--por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como un estándar. Puede adicionar un pasaje de hasta cinco palabras como un Texto de Portada, y un pasaje de hasta 25 palabras como un texto de Contra Portada, al final de la lista de Textos de Cubierta en la Versión Modificada. Solamente un pasaje de Texto de Portada y un Texto de Contra Portada puede ser adicionado por (o a manera de arreglos hechos por) cualquier entidad. Si el Documento ya incluye un texto de cubierta para la misma cubierta, previamente adicionado por usted o por arreglo hecho por la misma entidad, a nombre de la cual usted actua, no puede adicionar otra; pero puede reemplazar el anterior, con permiso explícito de quien previamente publicó y agregó tal texto. El(los) autor(es) y quien(es) publica(n) el Documento no dan con esta Licencia permiso para usar sus nombres para publicidad o para asegurar o implicar aprobación de cualquier Versión Modificada.

6. COMBINANDO DOCUMENTOS Puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, y las liste como Secciones Invariantes de su trabajo combinado en la respectiva nota de licencia. El trabajo combinado necesita contener solamente una copia de esta Licencia, y múltiples Secciones Invariantes idénticas pueden ser reemplazadas por una sola copia. Si hay múltiples Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único adicionándole al final de este, entre paréntesis, el nombre del autor o de quien publicó originalmente esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes en la nota de licencia del trabajo combinado. En la combinación, debe combinar cualquier sección titulada "Historia" de los varios documentos originales, formando una sección titulada "Historia"; de la misma forma combine cualquier seción titulada "Agradecimientos", y cualquier sección titulada "Dedicatorias". Debe borrar todas las secciones tituladas "Aprobaciones."

7. COLECCIONES DE DOCUMENTOS Usted puede hacer una colección que consista del Documento y otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en los

varios documentos con una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para una copia literal de cada uno de los documentos en cualquiera de todos los aspectos. Usted puede extraer un solo documento de una de tales colecciones, y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraido, y siga esta Licencia en todos los otros aspectos concernientes a la copia literal de tal documento.

8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES Una recopilación del Documento o de sus derivados con otros documentos o trabajos separados e independientes, en cualquier tipo de distribución o medio de almacenamiento, no cuenta como un todo como una Versión Modificada del Documento, siempre que no se clamen derechos de reproducción por la compilación. Tal recopilación es llamada un "agregado", y esta Licencia no aplica a los otros trabajos auto-contenidos y por lo tanto compilados con el Documento, o a cuenta de haber sido compilados, si no son ellos mismos trabajos derivados del Documento. Si el requerimiento de la sección 3 del Texto de la Cubierta es aplicable a estas copias del Documento, entonces si el Documento es menor que un cuarto del agregado entero, Los Textos de la Cubierta del Documento pueden ser colocados en cubiertas que enmarquen solamente el Documento entre el agregado. De otra forma deben aparecer en cubiertas enmarcando todo el agregado.

9. TRADUCCIÓN La traducción es considerada como una clase de modificación, así que puede distribuir traducciones del Documento bajo los términos de la sección 4. Reemplazar las Secciones Invariantes con traducciones requiere permiso especial de los propietarios de los derechos de reproducción, pero usted puede incluir traducciones de algunas o todas las Secciones Invariantes además de las versiones originales de las Secciones Invariantes. Puede incluir una traducción de esta Licencia siempre que incluya también la versión original en ingles de esta Licencia. En caso de un desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la versión original en Inglés prevalecerá.

10. TERMINACIÓN Usted no puede copiar, modificar, sublicenciar, o distribuir el Documento excepto como lo permite expresamente esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o distribución del Documento es nulo, y terminarán automáticamente sus derechos bajo esta Licencia. Sin embargo, los terceros que hayan recibido copias, o derechos, de su parte bajo esta Licencia no tendrán por terminadas sus licencias siempre que tales terceros permenezcan en total conformidad.

11. REVISIONES FUTURAS DE ESTA LICENCIA La Free Software Foundation puede publicar nuevas y revisadas versiones de la GNU Free Documentation License de tiempo en tiempo. Tales versiones nuevas

serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar problemas o intereses. Vea http://www.gnu.org/copyleft/. Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que una versión numerada particularmente de esta licencia o "cualquier versión posterior" se aplica a este, tiene la opción de seguir los términos y condiciones de esa versión especificada o de cualquiera versión posterior que hubiera sido publicada (no como un borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada(no como un borrador) por la Free Software Foundation.