

# GUÍA DE CVS

**(Concurrent Versions System)**

Proyecto de Ingeniería de Software – Curso 2001

## **INDICE**

<b>1. INTRODUCCIÓN .....</b>	<b>3</b>
1.1. Una sesión simple.....	3
<b>2. EL REPOSITORIO .....</b>	<b>5</b>
<b>3. INICIAR UN PROYECTO CON CVS.....</b>	<b>6</b>
3.1. Creación del árbol de directorios desde cero.....	6
3.2. Definición de módulos .....	7
<b>4. REVISIONES .....</b>	<b>8</b>
4.1. Números de Revisiones automáticos .....	8
4.2. Números de Revisiones asignadas .....	9
4.3. Revisiones simbólicas con marcas (tags) .....	9
<b>5. RAMIFICACIONES Y COMBINACIONES (BRANCHING&amp;MERGING) .....</b>	<b>11</b>
5.1. Creación de ramificaciones (branches) .....	11
5.2. Acceso a las ramificaciones .....	11
5.3. Ramificaciones y revisiones.....	13
5.4. Combinación (merge) de ramificaciones .....	13
<b>6. COMPORTAMIENTO RECURSIVO .....</b>	<b>15</b>
<b>7. AGREGAR, ELIMINAR Y RENOMBRAR ARCHIVOS Y DIRECTORIOS .....</b>	<b>15</b>
7.1. Agregar archivos y directorios .....	15
7.2. Eliminar archivos y directorios.....	16
7.3. Mover y renombrar archivos y directorios .....	16
<b>8. VER LA HISTORIA.....</b>	<b>17</b>
<b>9. MANEJO DE ARCHIVOS BINARIOS.....</b>	<b>18</b>
<b>10. MÚLTIPLES DESARROLLADORES.....</b>	<b>19</b>
10.1. Estado de los archivos.....	19
10.2. Mecanismos para seguimiento de archivos .....	21
10.3. Recuperación reservada o no reservada .....	23
<b>11. GESTIÓN DE REVISIONES .....</b>	<b>24</b>
<b>12. APÉNDICE - REFERENCIA RÁPIDA DE COMANDOS CVS .....</b>	<b>24</b>
<b>13. REFERENCIAS.....</b>	<b>28</b>

## 1. Introducción

CVS es un sistema de control de versiones mediante el cual se puede registrar la historia de los archivos fuente generados durante un proyecto de software. CVS almacena *todas* las versiones de un archivo en un *único* archivo guardando solamente las diferencias entre las distintas versiones, lo cual permite ahorrar espacio en disco. También ayuda a prevenir que los cambios realizados por distintas personas sean sobrescritos, creando un directorio de trabajo propio para cada persona y combinando el trabajo una vez que cada una ha terminado.

Sin embargo, CVS no es un sustituto de la gestión de configuración de un proyecto, es un instrumento de apoyo para realizarla, la persona responsable de la gestión de configuración es quién debe definir y controlar dicha gestión. CVS tampoco indica como debe construirse el software, simplemente almacena los archivos en la estructura definida permitiendo su recuperación. Descubrir que archivos se deben reconstruir cuando ocurre un cambio, está fuera del alcance de CVS. Lo mismo ocurre cuando se realizan cambios simultáneos sobre un mismo archivo, o a través de una colección de archivos, que determinen la aparición de conflictos lógicos entre ellos, por ejemplo al cambiar los argumentos en una función.

**Nota:** en esta guía se encuentran resumidos los conceptos básicos para el manejo de la herramienta CVS, dejando los detalles que tienen que ver con su uso específico en el curso 2001 de la Asignatura Proyecto de Ingeniería de Software en un documento aparte. De todas formas en algunas secciones se hace referencia a algunos aspectos que tienen que ver con el curso 2001.

Para tener una primera idea del funcionamiento de CVS se detallará en lo que sigue, una sesión de trabajo típica con CVS:

### 1.1. Una sesión simple

El primer punto a tener en cuenta es que CVS almacena todos los archivos en un *repositorio* centralizado que será detallado en la Sección 2 de esta guía, y que en esta sección se considera creado con nombre 'proyecto'.

Se tiene un código que consiste de varios archivos en C y un Makefile y se supone que en el repositorio se creó un módulo de nombre 'tc' (por 'trivial compiler').

Lo primero que se debe hacer es conseguir una copia de trabajo propia del código de 'tc', para lo cual se utiliza el comando *checkout*:

**\$ cvs checkout tc**

que creará un nuevo directorio de nombre 'tc' y lo cargará con los archivos de código correspondientes, que se pueden ver haciendo:

**\$ cd tc**

**\$ ls**

CVS      makefile      backend.c      driver.c      frontend.c      parser.c

El directorio CVS es utilizado internamente por CVS y NO se deben modificar o borrar ninguno de los archivos que contiene.

Una vez obtenida la copia de trabajo propia del código de 'tc', se realizarán los cambios en los archivos necesarios y se registrará una nueva versión de los mismos utilizando el comando *commit*:

**\$ cvs commit backend.c**

lo cual almacenará la nueva copia del archivo backend.c en el repositorio dejándola disponible para cualquier otra persona que esté utilizando el mismo repositorio.

CVS abre un editor de texto que permite ingresar un mensaje de log para el archivo como por ejemplo "con optimización", luego se salva el archivo temporal y se sale del editor. La variable de ambiente \$CVSEDITOR determina el editor que se inicia y si no está seteada se utilizará la variable de ambiente \$EDITOR si está seteada, si ninguna de las dos está seteada se utilizará la opción por defecto que varía según el sistema operativo, por ejemplo en UNIX es el vi y en Windows NT/95/98 el notepad. Si no se desea iniciar un editor se puede especificar el mensaje de log para el archivo desde la línea de comando utilizando la bandera -m de la siguiente forma:

**\$ cvs commit -m "con optimización" backend.c**

Al finalizar las tareas se elimina la copia de trabajo de tc, para lo cual se debe utilizar la forma de hacerlo de CVS, que es utilizando el comando *release* el cual chequea que se haya hecho *commit* de todas las modificaciones realizadas haciendo una anotación en la historia del archivo y utilizando la bandera -d para eliminar la copia de trabajo, como en el siguiente ejemplo:

```
$ cd ..
$ cvs release -d tc
M driver.c
? tc
You have [1] altered files in this repository.
Are you sure you want to release (and delete) module 'tc': n
** 'release' aborted by user choice.
```

En este caso hay dos advertencias por parte de CVS. La primera '? tc' significa que el archivo 'tc' es desconocido para CVS, lo cual puede ocurrir si por ejemplo, tc es el ejecutable y no estaba almacenado en el repositorio. También podría ser un archivo nuevo creado en la sesión de trabajo por lo tanto CVS no lo reconoce. Si se desea poner el archivo bajo control de configuración se utiliza el comando *add* para agregarlo al repositorio:

```
$ cd tc
$ cvs add tc
```

Si es un archivo binario se utiliza la bandera '- kb' que se explica posteriormente en la sección 9 – Manejo de archivos binarios. Los archivos agregados no son almacenados en el repositorio hasta que se realice *commit* para hacer el cambio permanente.

La segunda advertencia indicada por 'M driver.c' significa que el archivo ha sido modificado desde que fue recuperado. El comando *release* siempre finaliza indicando la cantidad de archivos modificados que hay en la copia de trabajo, pidiendo confirmación antes de borrar cualquier archivo o realizar cualquier anotación en la historia del archivo. En el ejemplo se respondió que No y la operación de *release* fue abortada. Para ver que sucedió con el archivo driver.c se utiliza el comando *diff* que compara la versión que fue recuperada con la que se encuentra en la copia de trabajo:

```
$ cd tc
$ cvs diff driver.c
```

La salida muestra las diferencias entre los archivos comparados, en este caso se agregó una línea para habilitar la optimización realizada, por lo que se realiza *commit* del archivo y luego *release* del módulo:

**\$ cvs commit -m "con optimización" driver.c**

```

Checking in driver.c;
/usr/local/proyecto/tc/driver.c, v ← driver.c
new revision: 1.2; previous revision: 1.1
done
$ cd ..
$ cvs release -d tc
? tc
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module 'tc': y

```

En el ejemplo no se hizo commit del add de tc por lo que la advertencia '?tc' aparece nuevamente pero es ignorada ya que no interesa agregar ese archivo al repositorio, se contesta Yes y CVS elimina la copia de trabajo propia.

*Fin del ejemplo.*

## 2. El Repositorio

El repositorio de CVS almacena una copia completa de los archivos y directorios que están bajo control de configuración. Generalmente no se accede a ninguno de los archivos del repositorio directamente sino que se utilizan los comandos de CVS para obtener una copia de los archivos en un directorio de trabajo propio, trabajando luego sobre esa copia y devolviéndolos al repositorio una vez finalizado el trabajo sobre los archivos.

La estructura general del repositorio es un árbol de directorios que se corresponde con los directorios en el directorio de trabajo, que contienen archivos de historia para cada archivo bajo control de configuración, identificados por el nombre del archivo con ',v' al final. Por ejemplo suponiendo que el repositorio se encuentra definido en /usr/local/proyecto, un posible árbol de directorios sería:

```

/usr
|
+--local
|
+--proyecto
|
+--CVSROOT (archivos administrativos de CVS)
|
+--prueba
|
+--tc
|
+-- Makefile, v
+-- backend.c, v
+-- driver.c, v
+-- frontend.c, v
+-- parser.c, v
|
+--man
|
+-- tc.1, v
|
+--testing
|
+-- testpgm.t, v
+-- test2.t, v

```

Los archivos de historia contienen entre otras cosas, la información suficiente para recrear cualquier revisión del archivo, un log de todos los mensajes de *commit* y el nombre del usuario que realizó *commit* de cada revisión. Los archivos de historia se conocen como archivos RCS porque el primer programa en almacenar archivos en ese formato fue un sistema de control de versiones conocido como RCS.

Todos los archivos *\*,v* de historia son read-only y no deben cambiarse estos permisos. Los directorios dentro del repositorio podrán ser escritos por las personas que tengan permiso para modificar los archivos en cada directorio. Algunas veces CVS puede almacenar archivos en el llamado Attic, lo cual desde el punto de vista del usuario no es importante, ya que CVS tiene esto en cuenta y lo revisa cuando es necesario.

En los directorios de trabajo se crea un directorio CVS que contiene varios archivos que son manejados por CVS, con información como por ejemplo, donde se ubica el repositorio, con qué directorio en el repositorio se corresponde el directorio de trabajo, los archivos y directorios que hay en el directorio de trabajo, entre otros.

En el directorio CVSROOT del repositorio se almacenan los archivos administrativos, los cuales si bien no son necesarios, hacen que algunos comandos funcionen mejor cuando por lo menos el archivo *'modules'* está seteado apropiadamente. Este archivo define todos los módulos en el repositorio siendo su estructura orientada a líneas. En su forma más simple, cada línea contiene el nombre del módulo, espacio en blanco y el directorio (relativo a la raíz del repositorio) donde reside el módulo, por ejemplo:

```
tc      prueba/tc
```

Para editar los archivos administrativos se debe recuperar una copia de trabajo con *'cvs checkout CVSROOT'*, editar los que corresponda y realizar *commit* de los cambios normalmente. Se debe tener cuidado de no hacer *commit* erróneo de un archivo administrativo porque puede suceder que un error en un archivo administrativo haga imposible hacer *commit* de nuevas revisiones.

**Nota:** A los efectos del curso 2001 cada grupo de proyecto tendrá un repositorio creado en una maquina de facultad, al que podrán acceder mediante los usuarios del curso que serán entregados. Los detalles sobre los usuarios y las definiciones de trabajo para el curso se brindan en un documento aparte. De todas formas, el Responsable de la Gestion de Configuracion debera realizar las definiciones para el trabajo en su grupo, la creacion de la estructura de directorios en el repositorio y lo que considere necesario para el correcto uso de la herramienta y del repositorio por parte de todos los integrantes del grupo.

### 3. Iniciar un Proyecto con CVS

Lo primero que se debe hacer al iniciar un nuevo proyecto es pensar en la organización que se le dará a los archivos que se mantendrán bajo control de configuración. Para eso es recomendable definir el árbol de directorios que se construirá en el repositorio y la lógica del mismo, de forma que todos los integrantes del grupo puedan manejar las distintas versiones de los archivos.

#### 3.1. Creación del árbol de directorios desde cero

Una vez definida la estructura de directorios, lo más fácil es crear el árbol desde cero y luego importarlo al repositorio, como se ve en el siguiente ejemplo:

```
$ mkdir tc
$ mkdir tc/man
$ mkdir tc/testing
```

y luego se utiliza el comando *import* para crear la estructura (vacía) dentro del repositorio, moviéndose primero al directorio que será importado y realizando la operación de *import* desde ahí:

```
$ cd tc
```

```
$ cvs import -m "creación de la estructura de directorios" prueba/tc com fin
```

```
cvs import: Importing /usr/local/proyecto/prueba/tc
```

```
cvs import: Importing /usr/local/proyecto/prueba/tc/man
```

```
cvs import: Importing /usr/local/proyecto/prueba/tc/testing
```

```
No conflicts created by this import
```

siendo 'prueba/tc' el directorio que se creará bajo el repositorio 'proyecto', y 'com' y 'fin' dos tags (que pueden tener cualquier nombre) requeridos por el comando *import*, pero que en el caso de creación de estructuras no son de utilidad. Luego se podrán ir agregando otros directorios y archivos en la estructura definida, como se detalla en la sección 7 – Agregar, eliminar y renombrar archivos y directorios.

La estructura resultante dentro del repositorio se puede ver como:

```
/usr
|
+--local
|
+--proyecto
|
+--CVSROOT (archivos administrativos de CVS)
|
+--prueba (directorio creado vacío)
|
+--tc (directorio creado vacío)
|
+--man (directorio creado vacío)
|
+--testing (directorio creado vacío)
```

**Nota:** es conveniente que al comenzar el estudio de CVS se definan un directorio prueba (como en el ejemplo) dentro del repositorio, donde puedan definirse directorios y archivos para hacer las pruebas que necesiten sin afectar lo que será luego la estructura de directorios de trabajo.

### 3.2. Definición de módulos

Como ya se vio en la sección 2 – Repositorio, se pueden definir módulos en el archivo 'modules' de CVS, que aunque no es necesario puede ser conveniente para agrupar archivos y directorios relacionados. Para esto se debe recuperar una copia de trabajo propia del archivo 'módulos' de CVS con el comando checkout:

```
$ cvs checkout CVSROOT/modules
```

```
$ cd CVSROOT
```

y luego editar el archivo e insertar la línea que define al módulo:

```
tc prueba/tc
```

finalmente se realiza el commit de los cambios hechos al archivo 'modules':

```
$ cvs commit -m "módulo tc agregado" modules
```

```

Checking in modules;
/usr/local/proyecto/CVSROOT/modules,v <-- modules
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database

```

y se elimina la copia de trabajo propia de CVSROOT

```

$ cd ..
$ cvs release -d CVSROOT

```

**Nota:** Cuando se define un modulo dentro de la estructura de directorios del repositorio, CVS permite que el modulo sea recuperado sin sus directorios padres, o sea que se puede recuperar solamente el modulo y su contenido a un directorio de trabajo. Esto puede ser importante cuando varias personas estén trabajando a la vez pero sobre distintos módulos, para evitar superposiciones e inconvenientes que pueden aparecer cuando las copias de trabajo son sobre los mismos directorios.

## 4. Revisiones

En general no debe ser una preocupación la asignación de números de revisión a los archivos ya que CVS asigna números en forma automática a partir de 1.1, 1.2 y así sucesivamente. Sin embargo si alguien quisiera tener mayor conocimiento y control sobre la asignación de números de revisión existe la forma de hacerlo. También es posible hacer el seguimiento de un conjunto de revisiones que involucren a más de un archivo, por ejemplo para saber que revisiones se corresponden con un release (liberación) del software determinada se utiliza un tag para marcarla, que es una revisión simbólica que se asigna a una revisión numérica en cada archivo correspondiente.

Conceptos que utiliza CVS: Un archivo puede tener varias versiones, así como también puede tenerlas un producto de software. Una versión en el primer sentido, es decir para un archivo, se nombrará como **revisión** y una versión en el segundo sentido, es decir para un producto de software compuesto de varios archivos, se nombrará como **release** o **liberación** (en castellano para no confundir con el comando *release* de CVS)

### 4.1. Números de Revisiones automáticos

Cada versión de un archivo, o sea cada revisión, tiene un único número de revisión. Estos números son de la forma 1.1, 1.2, 1.3.2.2, incluso 1.3.2.2.4.5. Un número de revisión está siempre compuesto por un número par de enteros decimales periódicamente separados, siendo por defecto la revisión 1.1 la primera asignada a un archivo. A cada revisión sucesiva se le asigna un nuevo número incrementando en uno el de más a la derecha.

```

+-----+   +-----+   +-----+   +-----+   +-----+
| 1.1 |-----| 1.2 |-----| 1.3 |-----| 1.4 |-----| 1.5 |
+-----+   +-----+   +-----+   +-----+   +-----+

```

Un número de revisión como el 1.3.2.2 representa una revisión en una ramificación lo cual se explica en detalle en la sección 5 – Ramificaciones y combinaciones, siendo en este caso la ramificación 1.3.2.

Cuando se agrega un nuevo archivo, el segundo número será siempre 1 mientras que el primero será igual al mayor primer número de cualquier archivo en el directorio. Por ejemplo, si se tienen los números de revisión 1.7.3.1 y 4.12, al agregar un nuevo archivo se le dará el número de revisión 4.1.



## 4.2. Números de Revisiones asignadas

En general es más fácil tomar los números de revisión asignados por CVS y utilizar las marcas (tags) para distinguir las distintas liberaciones de productos, sin embargo si se desea asignar un número de revisión se utiliza la opción `-r` con el comando *commit*, de la siguiente forma:

```
$ cvs commit -r 3.0
```

La opción `-r` implica la opción `-f` que causa que se realice commit de los archivos incluso si estos no fueron modificados. El número especificado con `-r` debe ser mayor que cualquier número de revisión existente, esto es si la revisión 3.0 existe no es posible realizar 'cvs commit -r 1.3'.

## 4.3. Revisiones simbólicas con marcas (tags)

Los números de revisiones tienen vida propia y no necesariamente tienen que ver con los números de liberación del software. Dependiendo de cómo se utiliza CVS, los números de revisión pueden cambiar varias veces entre dos liberaciones. En general no hay razón para ponerle una marca (tag) a un solo archivo, el uso más común es ponerle una marca a todos los archivos que componen un módulo determinado en puntos estratégicos del ciclo de vida del desarrollo, como al realizar una liberación. Para esto se utiliza el comando *cvs tag* una vez que se ha recuperado una copia de trabajo propia del módulo, como en el siguiente ejemplo:

```
$ cd tc
$ cvs tag rel-1-0
cvs tag: Tagging
T Makefile
T backend.c
T driver.c
T frontend.c
T parser.c
```

**Nota:** Cuando se le da a CVS como argumento un directorio, generalmente aplica la operación a todos los archivos en ese directorio y recursivamente a todos los directorios que pudiera contener, lo cual se detalla en la sección 6 – Comportamiento recursivo.

Los nombres de las marcas deben comenzar con una letra (mayúscula o minúscula) y pueden tener letras (también en mayúscula o minúscula), dígitos, '\_' y '-'.

Es importante definir una convención para los nombres de las marcas como por ejemplo, el nombre del programa y el número de versión de la liberación cambiando el '.' por '-' para evitar confusiones, con lo que CVS 1.9 se nombraría como CVS1-9.

Para ver todas las marcas que tiene un archivo y los números de revisión que representan, se utiliza el comando *status* con la bandera `-v`:

```
$ cvs status -v backend.c
```

```
=====
File: backend.c                               Status: Up-to-date

Working revision:    1.4                      Wed Aug 22 01:24:30 2001
Repository revision: 1.4                      /usr/local/proyecto/prueba/tc/backend.c,v
Sticky Tag:          (none)
Sticky Date:         (none)
Sticky Options:      (none)

Existing Tags:
rel-1-0                                (revision: 1.4)
```

El comando *checkout* con la bandera *-r* permite recuperar una cierta revisión de un módulo, por ejemplo para recuperar las fuentes que componen la liberación 1.0 del módulo 'tc' en cualquier momento:

### \$ cvs checkout -r rel-1-0 tc

lo que puede ser útil si por ejemplo alguien asegura que hay un bug en esa liberación pero no se encuentra el bug en la copia de trabajo actual. También se puede recuperar un módulo según determinada fecha en la misma forma, lo que puede verse en las opciones del comando *checkout* en la sección 12 - Apéndice – Referencia rápida de comandos CVS.

Cuando se marca más de un archivo con la misma marca (tag) se puede pensar en la marca como una curva dibujada en una matriz de nombre de archivo vs. número de revisión, por ejemplo si se tienen los siguientes archivos:

arch1	arch2	arch3	arch4	arch5	
1.1	1.1	1.1	1.1	1.1	/-- 1.1* <-* TAG
1.2	*-	1.2	1.2	- 1.2*-	
1.3	-- 1.3*-	1.3	/	1.3	
1.4		\	1.4	/	1.4
		\-	1.5 *-	1.5	
			1.6		

donde en algún momento en el pasado las versiones \* fueron marcadas con el mismo tag, obteniéndose las revisiones que componen la marca.

**Nota:** a los efectos del curso 2001, los tags se deberán utilizar para marcar las liberaciones de productos por iteración dentro de cada Fase, o sea que como mínimo se tendrán dos liberaciones en cada Fase. De todas formas el grupo podrá definir utilizar tags para realizar otras identificaciones (marcas) de productos a su conveniencia.

### Marcas persistentes (Sticky tags)

Algunas veces una copia de trabajo puede tener datos extra asociados, por ejemplo si pertenece a alguna ramificación o si está restringida a versiones previas a cierta fecha. Como estos datos persisten, o sea se aplican en los sucesivos comandos que se utilicen en la copia de trabajo, se conocen como "sticky". Estas marcas persisten en los archivos de la copia de trabajo hasta que son borrados utilizando el comando *update* con la bandera *-A* que recupera la última versión del archivo en el tronco principal, olvidando cualquier marca persistente, de fecha o de opción.

El uso más común de las marcas persistentes es identificar en que ramificación se está trabajando, como se describe posteriormente en la sección 5 – Ramificaciones y combinaciones. También tienen otros usos, por ejemplo, si se quiere prevenir de actualizar la copia de trabajo para aislarse de posibles cambios desestabilizadores que estén realizando otras personas. Obviamente esto puede hacerse no ejecutando el comando *update*, pero si se trata solo de una porción de un árbol más grande se pueden utilizar para recuperar una cierta revisión que quedará como marca persistente con lo cual la ejecución de comandos *update* no recuperará la última versión hasta que no se elimine la marca persistente con *update -A*. De la misma forma se puede utilizar una determinada fecha al recuperar una versión con la opción *-D*.

## 5. Ramificaciones y combinaciones (branching&merging)

CVS permite aislar cambios en una línea separada de desarrollo, lo que se conoce como ramificación. Cuando se realizan cambios en una ramificación, estos cambios no aparecen en el tronco principal o en otras ramificaciones. Más tarde se pueden mover cambios de una ramificación a otra o al tronco principal combinándolos. La combinación involucra la ejecución del comando *update -j* (por join) para combinar los cambios en el directorio de trabajo y luego realizar *commit* de esa revisión y efectivamente copiar esos cambios a otra ramificación.

Utilización: las ramificaciones son útiles, por ejemplo en el siguiente caso: se tiene la liberación 1.0 de 'tc' y se continúa el desarrollo para obtener la liberación 1.1 en unas semanas. En algún momento se descubre un bug en la liberación 1.0 y al revisarla se encuentra el problema, sin embargo, la revisión actual de los fuentes no está estable por lo que no se puede realizar una liberación basada en los nuevos fuentes. Para esto se puede crear una ramificación en cada árbol de revisión de todos los archivos que componen la liberación 1.0 de 'tc', y realizar las modificaciones en la ramificación sin alterar el desarrollo del tronco principal. Cuando se finalizan los cambios, se pueden incorporar en el tronco principal o se pueden dejar en la ramificación.

### 5.1. Creación de ramificaciones (branches)

Para crear una ramificación se utiliza el comando *tag* con la bandera *-b* una vez que se ha recuperado una copia de trabajo propia del módulo, como sigue:

```
$ cd tc
$ cvs tag -b rel-1-0-cont
```

lo que abre una ramificación basada en las revisiones actuales que se encuentran en el directorio de trabajo, asignándole el nombre *rel-1-0-cont*. Es importante tener en cuenta que las ramificaciones se crean en el repositorio no en la copia de trabajo, y esta creación no implica que se cambie automáticamente a esa ramificación en la copia de trabajo.

También puede crearse una ramificación sin referenciar ninguna copia de trabajo, utilizando el comando *rtag*, como se muestra a continuación:

```
$ cvs rtag -b -r rel-1-0 rel-1-0-cont tc
```

lo que significa que esta ramificación se creará en la raíz de la revisión que se corresponde con la marca 'rel-1-0', que no necesita ser la más reciente ya que en general lo más útil es abrir una ramificación de una revisión anterior que se sabe estable.

Como en las revisiones simbólicas con marcas (tags), las ramificaciones pueden corresponderse con distintas revisiones de los archivos que componen la liberación sobre la cual se abre la ramificación, lo que se detalla en la sección 5.3 – Ramificaciones y revisiones.

### 5.2. Acceso a las ramificaciones

Hay dos formas de acceder a las ramificaciones: recuperarla directamente desde el repositorio o cambiar una copia de trabajo existente para que se ubique sobre la ramificación.

En el primer caso, se utiliza la bandera *-r* con el comando *checkout* seguido del nombre de la marca dado a la ramificación, por ejemplo:

```
$ cvs checkout -r rel-1-0-cont tc
```

lo que recupera la ramificación marcada como rel-1-0-cont en el módulo tc, esto es todos los archivos que componen la ramificación del módulo tc.

Si ya se tiene recuperada una copia de trabajo, se puede cambiar para ubicarla sobre la ramificación utilizando el comando *update* con la bandera *-r*, como sigue:

```
$ cd tc
```

```
$ cvs update -r rel-1-0-cont
```

sin importar si la copia de trabajo se encontraba sobre el tronco principal o sobre alguna otra ramificación, de esta forma se ubicará sobre la ramificación nombrada, y en forma similar a la ejecución del comando *update* en forma normal, se combinarán los cambios que se hayan realizado notificando de los conflictos que pudieran ocurrir al hacerlo.

Una vez que la copia de trabajo se encuentra ligada a una ramificación en particular, los cambios que se realicen crearán nuevas revisiones en esa ramificación mientras el tronco principal y otras ramificaciones no son afectadas.

Para saber en que ramificación se encuentra una copia de trabajo se utiliza el comando *status* con la bandera *-v*, y en el campo nombrado como sticky tag, CVS indica en que ramificación se encuentran los archivos de la copia de trabajo actual (si se está en alguna), por ejemplo dentro de la copia de trabajo refiriéndose a los siguientes archivos:

```
$ cvs status -v backend.c driver.c
```

```
=====
File: backend.c                               Status: Up-to-date

Working revision: 1.4                         Sat Aug 25 19:17:54 2001
Repository revision: 1.4                     /usr/local/proyecto/prueba/tc/backend.c,v
Sticky Tag: rel-1-0-cont (branch: 1.4.2)
Sticky Date: (none)
Sticky Options: (none)

Existing Tags:
  rel-1-0-cont1                               (branch: 1.5.2)
  rel-1-0-cont                               (branch: 1.4.2)
  rel-1-0                                     (revision: 1.4)

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

Working revision: 1.2.2.1                     Wed Aug 22 00:15:47 2001
Repository revision: 1.2.2.1                 /usr/local/proyecto/prueba/tc/driver.c,v
Sticky Tag: rel-1-0-cont (branch: 1.2.2)
Sticky Date: (none)
Sticky Options: (none)

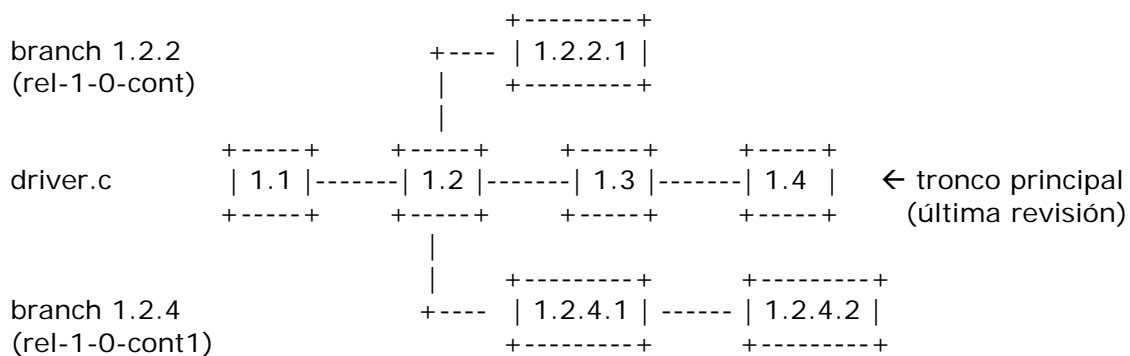
Existing Tags:
  rel-1-0-cont1                               (branch: 1.2.4)
  rel-1-0-cont                               (branch: 1.2.2)
  rel-1-0                                     (revision: 1.2)
```

Aclaraciones: no confundirse con el hecho de que los números de ramificación para cada archivo son distintos: backend.c tiene branch 1.4.2 y driver.c tiene branch 1.2.2, ya que la marca en ambas ramificaciones es la misma: rel-1-0-cont, los números simplemente reflejan el punto en la historia de revisiones de cada archivo en el cual se abrió la ramificación lo cual se detalla en la siguiente sección 5.3 – Ramificaciones y revisiones.

En el ejemplo, se puede deducir que backend.c sufrió más cambios que driver.c previo a la creación de la ramificación.

### 5.3. Ramificaciones y revisiones

Generalmente, la historia de revisiones de un archivo es una serie lineal de incrementos, como se indicó previamente. Sin embargo, como se vio en las secciones previas, CVS no está limitado a desarrollo lineal sino que el árbol de revisiones puede ser abierto en ramificaciones donde cada ramificación es una línea mantenida de desarrollo propio, donde los cambios realizados en una ramificación pueden ser movidos posteriormente al tronco principal. Cada ramificación tiene asignado un número de ramificación que está siempre compuesto por un número impar de enteros decimales periódicamente separados, que CVS forma agregando el primer número entero par no utilizado (comenzando con 2) al número de revisión donde la ramificación correspondiente se abre (se realiza el fork de la revisión). Esto permite que se pueda abrir más de una ramificación sobre una misma revisión. En una misma ramificación los números de revisión se asignan en la forma que se indicó en la sección 4.1 – Números de revisión automáticos. Para fijar ideas se muestra el árbol de revisiones del archivo driver.c de acuerdo a lo visto en la sección anterior:



también podría tenerse ramificaciones de ramificaciones, si en el ejemplo anterior se abriera una nueva ramificación a partir de la revisión 1.2.4.2 se tendría el número de branch 1.2.4.2.2, de acuerdo a la asignación de números de ramificación vista previamente.

### 5.4. Combinación (merge) de ramificaciones

Combinación de una ramificación entera:

Los cambios realizados en una ramificación pueden ser combinados en la copia de trabajo utilizando la bandera `-j branch` con el comando `update`, lo que produce que se combinen los cambios realizados entre el punto donde se abrió la ramificación y la revisión más nueva en esa ramificación, en la copia de trabajo. Utilizando el árbol de revisiones del archivo driver.c visto en la sección anterior, y asumiendo que existe un módulo mod que contiene solo ese archivo, se muestra como se haría esta combinación en el siguiente ejemplo:

```

$ checkout mod
$ cvs update -j rel-1-0-cont1 driver.c
$ cvs commit -m "incluyendo la ramificación rel-1-0-cont1"

```

en el cual primero se recupera la última revisión en el tronco principal, en este caso la revisión 1.4, luego se combinan todos los cambios realizados entre la revisión 1.2 y la revisión 1.2.4.2 en la copia de trabajo del archivo, y finalmente se crea la revisión 1.5.

Como resultado de la operación de combinación podrían aparecer conflictos, los cuales deberán ser resueltos antes de poder realizar el commit de la nueva revisión. En la sección 10 – Múltiples desarrolladores se detallan algunos ejemplos y sus posibles soluciones.

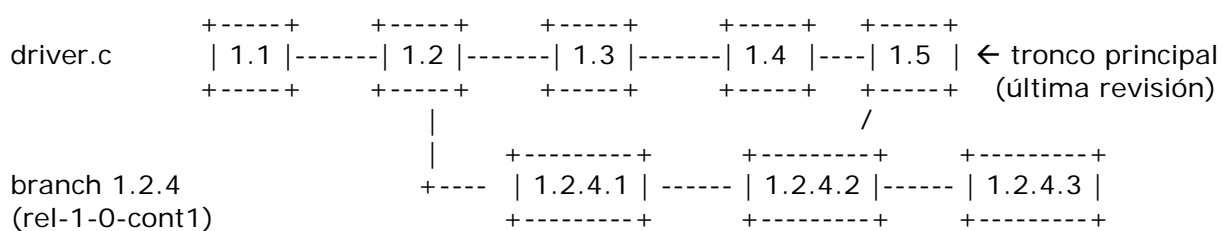
El comando checkout también soporta el uso de la bandera `-j branch`, por lo que el mismo efecto mostrado anteriormente podría alcanzarse utilizando:

**\$ cvs checkout -j rel-1-0-cont1 mod**

**\$ cvs commit -m "incluyendo la ramificación rel-1-0-cont1"**

Combinación de una ramificación más de una vez:

Luego de que se realizó la combinación anterior, el árbol de revisiones del archivo driver.c mostrando solo la ramificación de la combinación realizada, se encuentra de la siguiente forma:



suponiendo que el desarrollo continuó sobre la ramificación rel-1-0-cont1 y se tiene la revisión 1.2.4.3 del archivo en esa rama. Si se quiere combinar nuevamente la ramificación con el tronco principal utilizando el comando `update -j rel-1-0-cont1 driver.c` lo que intentará hacer CVS es combinar nuevamente los cambios que ya fueron combinados, desde la revisión 1.2 hasta la revisión 1.2.43, por lo que se debe especificar que los cambios a combinar son solamente los que aún no han sido combinados. Para eso se utiliza dos veces la opción `-j` indicando que se deben combinar los cambios desde la primera revisión indicada hasta la segunda revisión indicada, en el ejemplo:

**\$ cvs update -j 1.2.4.2 -j rel-1-0-cont1 driver.c**

que tiene el problema de especificar el número de revisión manualmente, algo un poco mejor sería utilizar la fecha en que se realizó la última combinación:

**\$ cvs update -j rel-1-0-cont1:yesterday -j rel-1-0-cont1 driver.c**

y lo mejor sería poner una marca simbólica cada vez que se realiza una combinación para poder utilizarla posteriormente para nuevas combinaciones:

**\$ cvs update -j combinada\_ramif\_con\_tronco -j rel-1-0-cont1 driver.c**

Combinación de diferencias entre dos revisiones cualesquiera:

Utilizando dos veces la opción `-j` con los comandos `update` y `checkout` se pueden combinar las diferencias entre dos revisiones en la copia de trabajo:

**\$ cvs update -j 1.5 -j 1.3 driver.c**    ¡ notar el orden en que se ponen las revisiones!

lo que produce que se eliminen los cambios hechos entre la revisión 1.3 y la revisión 1.5. Para utilizar esta opción sobre múltiples archivos se debe recordar que las revisiones numéricas probablemente sean distintas entre los archivos que componen un módulo,

por lo que es recomendable utilizar las marcas simbólicas de las liberaciones correspondientes.

## 6. Comportamiento recursivo

La mayoría de los comandos de CVS se realizan en forma recursiva cuando se especifica un directorio como argumento, por lo que por ejemplo, si `tc` es el directorio de trabajo actual y dentro del directorio `testing` se tienen los archivos `test1.t` y `test2.t` y se indica:

**\$ cvs update testing**

es equivalente a:

**\$ cvs update testing/test1.t testing/test2.t**

y si no se indican argumentos con el comando o se indica con `'.'`, es equivalente a aplicarlo sobre todos los archivos y directorios en el directorio de trabajo actual.

El comportamiento recursivo puede ser deshabilitado con la opción `-l`, que produce que no se aplique recursivamente sobre los directorios que contenga. Contrariamente, si como opción por defecto estuviera indicado `-l`, con el comando `-R` se puede forzar la recursión.

## 7. Agregar, eliminar y renombrar archivos y directorios

El concepto general a tener en cuenta cuando se agregan, eliminan o renombran archivos y directorios, es que en vez de realizar un cambio irreversible se quiere que CVS registre el hecho de que se ha realizado un cambio, de la misma forma que al modificar un archivo existente.

### 7.1. Agregar archivos y directorios

Para agregar archivos y directorios el mecanismo es el mismo pero la diferencia esta en la forma en que CVS lo registra. En ambos casos se debe tener una copia de trabajo del directorio donde serán creados, crear el nuevo archivo dentro de la copia de trabajo (o copiarlo/moverlo a la copia de trabajo) y finalmente utilizar el comando `add` para indicarle a CVS que lo debe incluir en el repositorio:

**\$ cvs add arch1**

**cvs add: scheduling file 'arch1' for addition**

**cvs add: use 'cvs commit' to add this file permanently**

lo que produce que CVS agende el archivo `arch1` para ser agregado al repositorio. Otros desarrolladores no podrán ver el archivo hasta que no se agregue en forma definitiva al repositorio mediante el comando `commit`. Al agregar directorios se debe tener en cuenta que a diferencia de la mayoría de los comandos, el comando `add` no es recursivo por lo que no se puede realizar por ejemplo, `cvs add modul1/arch1` sino que se debe agregar primero el directorio y luego el archivo.

**\$ cvs add modul1**

**Directory /usr/local/proyecto/prueba/modul1 added to the repository**

lo que a diferencia de los archivos, produce que CVS agregue el directorio `modul1` al repositorio sin necesidad de utilizar el comando `commit`.

Cuando se agrega un archivo en una copia de trabajo de una ramificación, solamente se agrega en esa ramificación, luego pueden combinarse las adiciones a otras ramificaciones o al tronco principal.

## 7.2. Eliminar archivos y directorios

Para eliminar un archivo y de todas formas poder recuperar revisiones anteriores, se elimina en el directorio de trabajo, luego se utiliza el comando *remove* para decirle a CVS que se quiere eliminar el archivo y finalmente se confirma la operación con el comando *commit*. Es importante al momento de eliminar un archivo verificar que no se han realizado cambios que no estén confirmados utilizando el comando *status* o *update*, ya que si se elimina con cambios no confirmados, no se podrá recuperar el archivo en la forma en que estaba al momento de eliminarlo.

Es posible que un archivo exista solamente en algunas ramificaciones y no en otras, o agregar posteriormente otro archivo con el mismo nombre. CVS creará o no el archivo basado en las opciones *-r* o *-D* especificadas con los comandos *checkout* o *update*.

```
$ cd testing
$ rm *.c
$ cvs remove
cvs remove: Removing
cvs remove: scheduling a.c for removing
cvs remove: scheduling b.c for removing
cvs remove: use 'cvs commit' to remove these files permanently
```

```
$ cvs commit -m "eliminacion de archivos"
cvs commit: Examining
cvs commit: Committing
```

lo que produce que se eliminen los archivos *.c* del directorio *testing*. Especificando la opción *-f* se puede eliminar en un solo paso, sin necesidad de hacer previamente *rm*.

```
$ cd testing
$ cvs remove -f *.c
```

si luego de eliminar un archivo con *remove* y antes de hacer *commit* se desea recuperar, se puede anular el *remove* utilizando el comando *add*, y si se desea recuperar luego de haber hecho *rm* sin haber hecho *remove*, se puede recuperar con el comando *update*. Igual que con el comando *add*, los archivos solo se eliminan en la copia de trabajo que se esté, si es una ramificación luego se pueden combinar las eliminaciones hacia otra ramificación o al tronco principal.

Para eliminar un directorio y luego poder recuperar versiones anteriores, el concepto es el mismo que para eliminar archivos, sin embargo, para eliminar un directorio se deben eliminar todos los archivos en él y no hay forma de eliminar el directorio en si mismo. Luego se utiliza la opción *-P* con *checkout* o *update* lo que produce que CVS elimine los directorios vacíos en la copia de trabajo. Esta opción está implícita en las opciones *-r* o *-D* de los comandos *checkout* y *update*, de forma que CVS pueda crear o no el directorio dependiendo de que la version en particular tenga o no archivos en ese directorio.

## 7.3. Mover y renombrar archivos y directorios

La forma común de mover un archivo es copiar el archivo *old* a *new* y luego utilizar los comandos CVS para eliminar el archivo *old* y agregar el archivo *new*.

```
$ mv old new
$ cvs remove old
$ cvs add new
$ cvs commit -m "renombrado old a new" old new
```



Esta es la forma más simple de mover un archivo y preservar la historia del mismo la cual se verá como historia de old hasta el momento del cambio y a partir de ahí como historia de new. Los números de revisión del archivo new comenzarán de nuevo, en general con la revisión 1.1, si esto no se desea se puede asignar un número de revisión como se vio en la sección 4.2 – Números de revisiones asignadas.

La forma común de renombrar o mover un directorio es renombrar o mover cada archivo en el directorio como se vio en la sección anterior, y luego al recuperar una copia de trabajo utilizar la opción -P para eliminar el directorio. Si se desea renombrar o eliminar un directorio del repositorio, se puede hacer de la siguiente manera:

- primero informar a todos quienes tengan una copia del módulo en el que será renombrado el directorio, para que puedan hacer commit de sus cambios y eliminar su copia de trabajo antes de que el directorio sea renombrado.
- Renombrar el directorio dentro del repositorio con:  
**\$ cvs \$CVSROOT/module**  
**\$ mv old-dir new-dir**
- reparar los archivos administrativos de CVS si es necesario, por ejemplo si se renombró un módulo entero.
- Avisar a todos que pueden recuperar el módulo y continuar trabajando.

Si alguien mantuvo una copia de trabajo los comandos CVS dejarán de funcionar hasta que no elimine el directorio que fue eliminado del repositorio. Siempre es mejor mover los archivos del directorio que el propio directorio, ya que posiblemente no se puedan recuperar en forma correcta antiguas versiones que probablemente dependan del nombre de los directorios.

## 8. Ver la historia

La historia del control de versiones guarda información sobre los archivos que han sido cambiados, cuando, como y por quién. Existen varios mecanismos para ver la historia, que se muestran a continuación:

### Mensajes de log:

Cada vez que se realiza un commit se especifica un mensaje de log, los cuales pueden verse utilizando el comando log de cvs, por ejemplo del archivo arch1 en sus primeras versiones:

```
$ cvs log arch1
RCS file: /usr/local/proyecto/prueba/modul1/arch1,v
Working file: arch1
head: 1.2
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;   selected revisions: 2
description:
-----
revision 1.2
date: 2000/08/30 01:34:09; author: user2; state: Exp; lines: +1 -0
cambiando lineas
-----
revision 1.1
date: 2000/08/28 01:17:13; author: user1; state: Exp;
creando archivo arch1
=====
```

Base de datos histórica:

CVS mantiene un archivo de historia que almacena el uso de los comandos CVS checkout, commit, rtag, update y release, para ver esta información se utiliza el comando history:

**\$ cvs history arch1**

Logging definido por el usuario:

Se puede indicar a CVS que haga log de varios tipos de acciones definidas de la forma que se elija, como por ejemplo enviar un mail a un grupo de desarrolladores, o poner un mensaje en determinado newsgroup. Estos mecanismos operan ejecutando un script, lo que requiere que estos se mantengan actualizados. Para hacer log de commits se utiliza el archivo loginfo, para hacer log de commits, checkouts, exports y tags, respectivamente se pueden utilizar también las opciones `-i`, `-o`, `-e` y `-t` en el archivo de módulos. Una forma menos trabajosa es utilizar el comando `watch add` el cual es útil incluso si no se está utilizando `cvs watch on`, lo cual será detallado en la sección 10 – Múltiples desarrolladores.

Comando *annotate*:

El comando `annotate` imprime la revisión que se encuentra en la cabeza del tronco principal (esto es la última), junto con información de la última modificación en cada línea:

**\$ cvs annotate arch1****Annotations for arch1**

\*\*\*\*\*

**1.1 (user1 27-Jul-00) : creando archivo arch1**

**1.2 (user2 28-Jul-00) : cambiando lineas**

en este ejemplo, el archivo `arch1` tiene solamente dos líneas, y este tipo de reporte no dice nada sobre líneas eliminadas o reemplazadas, para eso se debe utilizar el comando `diff`.

## 9. Manejo de archivos binarios

El uso más común de CVS es almacenar archivos de texto, con los cuales CVS puede combinar revisiones, desplegar las diferencias entre revisiones en forma visible a las personas y otras operaciones de ese tipo. De todas formas, si se desea se pueden almacenar archivos binarios, perdiendo algunas de estas características. Por ejemplo, se puede almacenar un sitio web incluyendo archivos de texto e imágenes binarias.

Una función básica del control de versiones es mostrar las diferencias entre dos revisiones, por ejemplo si alguien registró una nueva versión de un archivo, se puede ver que cambió mediante el comando `diff` y determinar si los cambios están bien. Con archivos binarios se pueden extraer dos revisiones y luego compararlos con alguna herramienta externa a CVS, si no la hubiera, se deben registrar los cambios con otros mecanismos como ser que se escriban buenos mensajes de log.

Otra habilidad importante en un sistema de control de versiones es la combinación de dos revisiones, lo cual CVS realiza en dos contextos: cuando los usuarios hacen cambios en copias de trabajo separadas y cuando se realiza una combinación explícitamente con el comando `update -j`. En el caso de archivos de texto, CVS puede combinar cambios realizados en forma separada y señalar un conflicto si los cambios presentaran conflictos. Con archivos binarios, lo máximo que puede hacer CVS es presentar las dos distintas copias del archivo al usuario y que este resuelva el conflicto, eligiendo un archivo o el otro.

Si el proceso de combinar archivos se ve como indeseable, la mejor elección es prevenir de hacer las combinaciones, en el caso de múltiples desarrolladores utilizando lockeo de archivos y en el caso de ramificaciones, restringiendo su uso.

Para almacenar archivos binarios se debe utilizar la opción `-kb` que previene la conversión de fin de línea y la expansión de teclas que normalmente utiliza CVS. También puede definirse un valor por defecto para que un archivo sea tratado como binario al agregarlo o importarlo con los comandos `add` e `import` de CVS, por ejemplo se puede especificar que los archivos cuyo nombre finalice en `'.exe'` son binarios.

## 10. Múltiples desarrolladores

Cuando varias personas trabajan en un proyecto, aparecen complicaciones cuando, por ejemplo, dos personas intentan editar el mismo archivo a la vez. Una solución conocida como lock de archivos o recuperación reservada permite que solo una persona edite el archivo en cada momento utilizando el comando CVS `admin -l`. También pueden utilizarse las características de la opción `watch` para prevenir que dos personas editen a la vez el mismo archivo. La recuperación por defecto en CVS es recuperación no reservada que permite que en cada momento los desarrolladores editen su copia de trabajo propia de un archivo en forma simultánea. La primer persona que realiza `commit` de sus cambios no tiene forma de saber que otra persona está cambiando el mismo archivo. El resto de las personas que esté trabajando sobre ese archivo obtendrá un error al momento de realizar `commit` de sus cambios. Para que su copia de trabajo pueda ingresarse al repositorio deberán utilizar comandos CVS que realizan este proceso en forma casi totalmente automática.

### 10.1. Estado de los archivos

Basado en las operaciones que se han realizado sobre un archivo recuperado y en las operaciones que otros han realizado sobre ese archivo en el repositorio, se puede clasificar un archivo en varios estados los que pueden verse con el comando `status`. Estos estados se describen a continuación:

**Up-to-date:** el archivo es idéntico a la última revisión en el repositorio para la ramificación en uso.

**Locally modified:** el archivo ha sido editado localmente pero no se ha realizado `commit` de los cambios.

**Locally added:** el archivo ha sido agregado localmente con el comando `add` pero no se ha realizado `commit` de los cambios.

**Locally removed:** el archivo ha sido eliminado localmente con el comando `remove` pero no se ha realizado `commit` de los cambios.

**Needs checkout:** otra persona ha realizado `commit` de una nueva versión en el repositorio. El nombre del estado puede conducir a malentendidos ya que en general es mejor utilizar el comando `update` que el `checkout` para recuperar la versión más nueva.

**Needs patch:** idem anterior pero será enviado un patch en lugar del archivo entero. Enviar un patch o enviar el archivo entero refleja el mismo problema.

**Needs merge:** otra persona ha realizado `commit` de una nueva versión en el repositorio, y quien recibe el mensaje también ha realizado cambios en el archivo en la copia de trabajo propia.

**File had conflicts on merge:** idem `locally modified` pero un comando `update` realizado previamente está dando conflictos. Los conflictos deben ser resueltos como se explica más adelante en esta sección.

**Unknown:** CVS no tiene información sobre el archivo, por ejemplo se ha creado un nuevo archivo pero no se ha agregado con el comando `add`.

El comando `status` también reporta otros estados como "Working revision" que es la revisión del archivo de la cual deriva la copia en el directorio de trabajo y "Repository revision" que es la última revisión en el repositorio para la ramificación en uso.

Los comandos `status` y `update` pueden verse como complementarios, el comando `update` se utiliza para poner un archivo en estado up-to-date y el comando `status` se puede utilizar para tener una idea de lo que hará el comando `update`.

### **Llevar un archivo al estado up-to-date:**

Para actualizar o combinar un archivo se utiliza el comando `update`, para archivos cuyo estado no es up-to-date esto es equivalente a ejecutar un comando `checkout`: la última revisión del archivo es recuperada del repositorio y puesta en la copia de trabajo propia del módulo. Las modificaciones realizadas a un archivo nunca se pierden utilizando el comando `update`, si no hay una revisión nueva simplemente no tiene efecto. Si se ha editado el archivo y hay una revisión nueva disponible, CVS combina los cambios en la copia de trabajo propia.

Por ejemplo, se ha recuperado la revisión 1.4 y se le han hecho cambios, en el medio, alguien ha realizado `commit` de la revisión 1.5 y un tiempo después de la revisión 1.6, por lo que al ejecutar `update` sobre el archivo, CVS incorporará todos los cambios entre la revisión 1.4 y la revisión 1.6 en su archivo. Si alguno de los cambios entre las revisiones 1.4 y 1.6 fueron hechos demasiado cerca ocurre una superposición y se imprime un aviso. El archivo resultante incluye las dos versiones de las líneas que se superponen delimitadas por marcas especiales, de forma que al editar el archivo se puedan identificar los conflictos, y luego eliminar las líneas erróneas conjuntamente con las marcas especiales. Como protección, CVS no permitirá ingresar un archivo que presenta conflictos que no fueron resueltos.

Por ejemplo, se supone definido el modulo `modul1` correspondiente al directorio creado en el repositorio bajo el directorio prueba, y recuperada una copia de trabajo propia del modulo `modul1`. Al mismo tiempo, otro integrante del grupo tiene recuperada una copia de trabajo del directorio prueba que contiene al directorio `modul1` y todos sus archivos y directorios. Supongamos que los dos trabajan sobre el mismo archivo `arch1` que se encuentra dentro del directorio `modul1`. El primero que termine su trabajo y haga `commit` de sus cambios, no tendrá ningún problema ni recibirá ningún mensaje de aviso de CVS, pero cuando la otra persona quiera realizar `commit` de sus cambios, CVS no lo permitirá porque el estado de la copia del archivo `arch1` en el directorio de trabajo no es up-to-date, es decir no coincide con la que está almacenada en el repositorio, y el siguiente aviso será enviado por CVS:

```
$ cvs commit -m "con cambios" arch1  
cvs commit: Up-to-date check failed for `arch1'  
cvs [commit aborted]: correct above errors first!
```

Entonces se deben combinar la copia del archivo existente en el directorio de trabajo con el archivo almacenado en el repositorio, para que pueda ser ingresado al mismo, utilizando el comando `update`:

```
$ cvs update arch1  
RCS file: /usr/local/proyecto/prueba/modul1/arch1,v  
retrieving revision 1.4  
retrieving revision 1.6  
Merging differences between 1.4 and 1.6 into arch1  
rcsmerge: warning: conflicts during merge  
cvs update: conflicts found in arch1  
C arch1
```

en este caso hay conflictos en el merge y se debe editar el archivo resultante para solucionarlos, para la identificación de los conflictos CVS utiliza marcas:

```
<<<<<< arch1
probando multiples
desarrolladores
=====
conflictos
en el merge
>>>>>> 1.6
esto lo tienen en comun
los dos archivos y no hay
problema en el merge
```

luego de solucionar los conflictos se eliminan las marcas de CVS en el archivo y se realiza commit del archivo en la copia de trabajo que contiene los cambios combinados del archivo almacenado en el repositorio con el archivo de trabajo y la solución de conflictos realizada al editar el archivo, y se genera una nueva revisión que se almacena en el repositorio:

```
$ cvs commit -m "arreglados los conflictos" arch1
Checking in arch1;
/usr/local/proyecto/prueba/modul1/arch1,v <-- arch1
new revision: 1.7 ; previous revision: 1.6
done
```

## 10.2. Mecanismos para seguimiento de archivos

En general para muchos grupos de desarrollo el uso de CVS en su modo por defecto está bien. Algunas veces al querer ingresar un archivo modificado al repositorio se encontrará que ya ha sido modificado y se resolverá el problema con los comandos CVS que se vieron previamente. Otros grupos prefieren saber quién está editando que archivos para que si dos personas intentan editar el mismo archivo puedan comunicarse y hablar sobre los cambios que harán en vez de sorprenderse luego al querer ingresar el archivo nuevamente al repositorio. Para el seguimiento de archivos se utilizan las características del comando `watch`, que se detallan a continuación.

**Nota:** Es recomendable que se utilice el comando `edit` (no el `chmod`) para hacer que los archivos sean read-write para editarlos y que se utilice el comando `release` (no `rm`) para eliminar un directorio de trabajo que ya no se usará.

### Opciones de watch:

Para habilitar las características de `watch` primero se especifica que archivos serán seguidos con el comando `watch on`, el cual especifica que los desarrolladores deberán ejecutar el comando `edit` antes de editar los archivos indicados. Los archivos serán recuperados como read-only por lo que los permisos deberán ser cambiados antes de trabajar sobre el archivo. Si se indica un directorio, todos los archivos y directorios en él serán seguidos, de acuerdo a lo visto en la sección 6 – comportamiento recursivo, poniendo la opción por defecto para archivos agregados en el futuro. Si no se indica nada se toma el directorio actual.

**cvs watch on [-IR] files ..**

Para que no se provea notificación del trabajo en determinados archivos se utiliza el comando `watch off` que creará copias de trabajo read-write en los archivos especificados, procesando las opciones en la misma forma que en `watch on`.

**cvs watch off [-IR] files ..**

Para que CVS envíe notificaciones de determinadas acciones realizadas sobre un archivo, se utiliza el comando *watch add* que no necesita que se haya especificado el comando *watch on* previamente, aunque esto será deseable para que se utilice el comando *edit*. El comando *watch add* agrega al usuario actual a la lista de personas que recibirán notificación sobre el trabajo que se realice en los archivos indicados:

#### **cvswatch add [-a actions] [-IR] files ..**

siendo las acciones que se pueden indicar las siguientes:

edit: otra persona ha ejecutado el comando *edit* sobre un archivo.

unedit: otra persona ha ejecutado el comando *unedit* sobre un archivo, o el comando *release*, o ha borrado un archivo y ejecutado *update* para recrearlo.

commit: otra persona ha realizado *commit* de los cambios en un archivo.

all: todos los anteriores.

none: ninguno de los anteriores.

Si se omite la opción *-a* por defecto se aplica la acción *all*. Las opciones y archivos se procesan igual que los anteriores comandos *watch*.

Para eliminar un pedido de notificación establecido con el comando anterior *watch add*, se utiliza el comando *watch remove* con los mismos argumentos. Si se especifica la opción *-a* entonces se eliminan solo las acciones especificadas.

#### **cvswatch remove [-a actions] [-IR] files ..**

Cuando se cumplen las condiciones de notificación, CVS hace una llamada al archivo administrativo '*notify*', el cual puede ser editado y agregarle por ejemplo, la siguiente línea: *ALL mail %s -s \"notificación CVS\"* para notificar vía e-mail a los usuarios especificados, o lo que se desee especificar. CVS también permite asociar una dirección de notificación establecida para cada usuario, creando el archivo '*users*' en '*CVSROOT*' con una línea por cada usuario en el formato *user:value*.

CVS no notifica a un usuario de sus propios cambios, ya que chequea que el usuario que dispara la notificación no sea el mismo que debe recibirla. En general, solo se sigue una edición por cada usuario.

#### Como editar un archivo que está bajo seguimiento:

Como los archivos bajo seguimiento tiene permiso *read-only*, no pueden simplemente editarse, sino que deben hacerse *read-write* e informar al resto de los desarrolladores que se planea editar el archivo, mediante el comando *edit*:

#### **cvswatch edit [options] files ..**

que acepta las mismas opciones que el *watch add* y procesa las opciones y files en la misma forma que el resto de los comandos *watch*. Generalmente luego de un conjunto de cambios, se realiza *commit* de los mismos lo que ingresa los cambios y regresa los archivos bajo seguimiento a su estado normal de *read-only*.

Pero si se decide abandonar los cambios o no hacer ningún cambio, se utiliza el comando *unedit* el cual produce que se abandone el trabajo sobre la copia del archivo y se vuelva a la versión del repositorio en la cual está basado el archivo:

#### **cvswatch unedit [-IR] files ..**

que procesa las opciones y files en la misma forma que el resto de los comandos *watch*.

CVS vuelve los archivos especificados a read-only para aquellos usuarios que han solicitado notificación mediante watch on y notifica a aquellos que hayan especificado notificación de unedit para los archivos especificados.

Si no hay seguimiento de archivos en uso, el comando unedit probablemente no funcione y la forma de volver a la versión en el repositorio es eliminar el archivo y utilizar el comando update para recuperar una nueva copia. El significado de esto no es el mismo, eliminar y actualizar puede introducir cambios que se hayan realizado en el repositorio desde la última vez que se utilizó el comando update.

#### Información sobre quién está haciendo seguimiento y editando:

Para listar los usuarios que actualmente están siguiendo los cambios sobre determinados archivos, se utiliza:

##### **cvs watchers [-IR] files ..**

que devuelve los archivos que están bajo seguimiento con información de cada una de las personas que siguen cada archivo.

Para listar los usuarios que actualmente están trabajando sobre determinados archivos, se utiliza:

##### **cvs editors [-IR] files ..**

que devuelve la dirección de e-mail de cada usuario, el momento en que cada usuario comenzó a trabajar con el archivo e información de ubicación del archivo.

En ambos casos las opciones y files son procesadas en la misma forma que el resto de los comandos watch.

### **10.3. Recuperación reservada o no reservada**

Esta es una elección que debe hacer el grupo de trabajo, por lo que en esta sección se describen brevemente las características de cada una para que se pueda realizar una elección adecuada a la forma de trabajo del grupo.

La recuperación reservada puede atentar contra la productividad: por ejemplo, si dos personas quieren editar distintas partes de un archivo, no habría razón para que no lo hicieran a la vez. También puede ocurrir que una persona tome un lock sobre un archivo y luego se olvide de liberarlo.

En general, trabajando con recuperación no reservada se ha observado que los conflictos en archivos ocurren raramente y son relativamente claros de resolver. Los conflictos serios en general ocurren cuando dos desarrolladores discrepan en el diseño apropiado para una sección de código, lo cual en realidad refleja que el equipo no se ha comunicado apropiadamente en primer lugar. Si se está de acuerdo en el diseño general del sistema, los cambios que se superponen son fáciles de combinar. En algunos casos, sin embargo, la recuperación no reservada es totalmente inapropiada, por ejemplo si no existen herramientas para combinar el tipo de archivos que se están manejando, la resolución de conflictos será demasiado problemática por lo que será mejor prevenirla utilizando recuperación reservada.

La opción de seguimiento de archivos es intermedia, ya que en lugar de prohibir que dos personas accedan al mismo archivo, permite saber quién más está editándolo y de acuerdo a la situación tomar la acción adecuada.

## 11. Gestión de revisiones

Hasta esta sección se describió el funcionamiento básico de CVS, pero hay cosas sobre el control de versiones que deben ser decididas por el grupo de trabajo, la más importante tiene que ver con el momento en que se realiza commit de los cambios.

Si el commit de los cambios en los archivos se realiza demasiado rápido, podría suceder que se ingresaran archivos que tal vez ni siquiera compilen, y cuando otra persona actualice su directorio de trabajo para incluir el archivo, no podrá compilar el código. Por otro lado, si se tarda demasiado en realizar el commit de los archivos, el resto del equipo no podrá beneficiarse de las mejoras realizadas al código y posiblemente los conflictos sean más comunes. En general se recomienda realizar commit de los archivos solamente después de que compilan.

## 12. Apéndice - Referencia rápida de comandos CVS

Un comando cvs tiene la siguiente forma:

**cvs [global\_options] command [command\_options] [command\_args]**

donde las opciones y argumentos básicos son:

### Global options:

- e *editor* edita mensajes con *editor*.
- H imprime un mensaje de ayuda.
- r hace que los nuevos archivos de trabajo sean read-only.
- v despliega información sobre la versión y copyright de CVS.
- w hace que los nuevos archivos de trabajo sean read-write.

### Commands, command options y command arguments:

#### **add [options] [files ..]**

agrega un nuevo archivo / directorio.

- m *msg* setea la descripción del archivo.

#### **annotate [options] [files ..]**

muestra la última revisión donde cada línea fue modificada.

- D *date* muestra la revisión más reciente no después de *date*.
- f utiliza la última revisión en el tronco principal (head) si no encuentra tag o date.
- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).
- r *tag* muestra la revisión identificada por *tag*.

#### **checkout [options] modules ..**

recupera una copia de las fuentes.

- A resetea cualquier sticky tag/date/options.
- D *date* recupera revisiones de la fecha *date* (sticky).
- d *dir* recupera en el directorio *dir*.
- f utiliza la última revisión en el tronco principal (head) si no encuentra tag o date.
- j *rev* combina los cambios.
- l local, opera solo en el directorio actual.
- n no ejecutar programa del módulo (si hubiera alguno)
- P elimina directorios vacíos.



- R opera recursivamente (opción por defecto).
- r *tag* recupera la revisión identificada por *tag* (sticky).

**commit [options] [files ..]**

confirma cambios en el repositorio.

- f obliga la confirmación del archivo, deshabilita la recursión.
- l local, opera solo en el directorio actual.
- m *msg* utiliza *msg* como mensaje de log.
- n no ejecutar programa del módulo (si hubiera alguno)
- R opera recursivamente (opción por defecto).
- r *rev* confirma como revisión *rev*.

**diff [options] [files ..]**

muestra las diferencias entre revisiones.

- D *date1* muestra las diferencias entre la revisión de fecha *date1* y la revisión en el directorio de trabajo.
- D *date2* muestra las diferencias entre la revisión *rev1/date1* y la revisión de fecha *date2*.
- l local, opera solo en el directorio actual.
- N incluye diferencias para archivos agregados y eliminados.
- R opera recursivamente (opción por defecto).
- r *rev1* muestra las diferencias entre la revisión *rev1* y la revisión en el directorio de trabajo.
- r *rev2* muestra las diferencias entre la revisión *rev1/date1* y la revisión *rev2*.

**edit [options] [files ..]**

prepara para editar un archivo que está siendo seguido.

- a *actions* especifica acciones para un seguimiento temporario, donde las acciones pueden ser: edit, unedit, commit, all y none.
- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).

**editors [options] [files ..]**

mira quién está editando un archivo que está siendo seguido.

- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).

**export [options] modules ..**

exporta archivos desde CVS.

- D *date* recupera revisiones de la fecha *date*.
- d *dir* recupera en el directorio *dir*.
- f utiliza la última revisión en el tronco principal (head) si no encuentra tag o date.
- l local, opera solo en el directorio actual.
- n no ejecutar programa del módulo (si hubiera alguno)
- P elimina directorios vacíos.
- R opera recursivamente (opción por defecto).
- r *tag* recupera la revisión identificada por *tag* (sticky).

**history [options] [files ..]**

muestra la historia de acceso al repositorio.

- a todos los usuarios (por defecto es "uno mismo").
- c reporta los archivos con modificaciones confirmadas (committed).

- D *date* muestra la historia desde *date*.
- e reporta todos los tipos de registro.
- l última modificación (reporte confirmado o modificado).
- m *module* reporta sobre módulo *module* (repetible).
- n *module* reporta en módulo *module*.
- o reporta los módulos recuperados.
- r *rev* reporta desde la revisión *rev*.
- T reporta sobre todas las marcas (TAGs).
- t *tag* reporta desde que se puso el registro *tag* en el archivo de historia (por cualquiera)
- u *user* para el usuario *user* (repetible).

**import [options] repository vendor-tag release-tag**

importa archivos hacia el repositorio, utilizando ramificaciones de "vendor".

- m *msg* utiliza *msg* como mensaje de log.

**log [options] [files ..]**

imprime información histórica para los archivos.

- b lista solo revisiones en la ramificación por defecto.
- d *date* especifica fechas (*d1* < *d2* para rango, *d* para la última antes).
- h solamente imprime el cabezal.
- l local, opera solo en el directorio actual.
- N no lista marcas (tags).
- R solo imprime el nombre del archivo RCS.
- r *revs* solo lista las revisiones *revs*.
- s *states* solo lista las revisiones en los estados *states*.
- t solo imprime el cabezal y texto descriptivo.

**rdiff [options] modules ..**

muestra diferencias entre liberaciones.

- D *date* selecciona revisiones basado en *date*.
- f utiliza la última revisión en el tronco principal (head) si no encuentra tag o date.
- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).
- r *rev* selecciona revisiones basado en *revs*.
- t las dos últimas diferencias entre los cambios hechos al archivo.

**release [options] directory ..**

indica que un directorio no está más en uso.

- d elimina el directorio especificado.

**remove [options] [files ..]**

elimina una entrada del repositorio.

- f borra el archivo antes de eliminarlo.
- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).

**rtag [options] tag modules ..**

agrega una marca simbólica a los módulos.

- a borra un tag de archivos eliminados que de otra forma quedarían marcados.
- b crea una ramificación de nombre *tag*.

- D *date* marca revisiones con *date*.
- d elimina el *tag* dado.
- f fuerza a que coincida la última revisión en el tronco principal (head) si no encuentra *tag* o *date*.
- l local, opera solo en el directorio actual.
- n no ejecución del programa marcado (*tag*).
- R opera recursivamente (opción por defecto).
- r *tag* marca el *tag* existente *tag*.

**status [options] files ..**

despliega información de estados en el directorio de trabajo.

- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).
- v incluye la información de marcas (tags) para el archivo.

**tag [options] tag [files ..]**

agrega una marca simbólica a las versiones recuperadas de los archivos.

- b crea una ramificación de nombre *tag*.
- D *date* marca revisiones con *date*.
- d elimina el *tag* dado.
- f fuerza a que coincida la última revisión en el tronco principal (head) si no encuentra *tag* o *date*.
- l local, opera solo en el directorio actual.
- n no ejecución del programa marcado (*tag*).
- R opera recursivamente (opción por defecto).
- r *tag* marca el *tag* existente *tag*.

**unedit [options] files ..**

deshace un comando edit.

- a actions especifica acciones para un seguimiento temporario, donde las acciones pueden ser: edit, unedit, commit, all y none.
- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).

**update [options] [files ..]**

sincroniza el directorio de trabajo con el repositorio.

- A resetea cualquier sticky tag/date/options.
- D *date* recupera revisiones de la fecha *date* (sticky).
- d crea directorios.
- f utiliza la última revisión en el tronco principal (head) si no encuentra *tag* o *date*.
- j *rev* combina los cambios.
- l local, opera solo en el directorio actual.
- P elimina directorios vacíos.
- R opera recursivamente (opción por defecto).
- r *tag* recupera la revisión identificada por *tag* (sticky).

**watch [on|off|add|remove] [options] [files ..]**

para hacer seguimiento de archivos.

on/off pone la recuperación de archivos como si/no en modo read-only.

Add/remove agrega o elimina notificaciones de acciones.

- a actions especifica acciones para un seguimiento temporario, donde las acciones pueden ser: edit, unedit, commit, all y none.
- l local, opera solo en el directorio actual.

- R opera recursivamente (opción por defecto).

**watchers [options] [files ..]**

mira quien está haciendo seguimiento de archivos.

- l local, opera solo en el directorio actual.
- R opera recursivamente (opción por defecto).

### 13. Referencias

El manual de "Per Cederqvist et al" sobre el que está basada esta guía se encuentra disponible en inglés en la dirección: <http://www.cvshome.org>, así como otra información de ayuda sobre CVS en FAQ.