CVS+ELVIRA

Índice General

1 ¿Qué es CVS? 2

2 ¿Qué no es CVS? 2

3 ¿Por qué CVS? 2

4 Conceptos Básicos 2

5 Primeros Pasos 3

6 Sesiones con CVS 4

6.1 Inicio de la Sesión 4
6.2 Obteniendo una Versión: Comando checkout 4
6.3 El directorio CVSROOT y otros 5
6.4 Editando y Compilando Ficheros Java 5
6.5 Actualizando Ficheros Modificados: Comando commit 5
6.5.1 Notificación Automática de Revisiones 8
6.6 Añadiendo Nuevos Ficheros: Comando add 9
6.7 Borrando Ficheros: Comando remove 10
6.8 Fusión automática de Ficheros: Comando update 12
6.9 Estado de los Ficheros: Comando status 14
6.10 Cerrando la Sesión y Borrando Directorio Local de Trabajo: Comando release 16

7 Resolución de Conflictos: Comando update 17

7.1 Ejemplo 1 17
7.2 Ejemplo 2 20
7.3 ¿Qué hacer en caso de Conflicto? 23

8 Visualizando BAYELVIRA por la WEB 24

9 Quiero saber más sobre CVS 25

1 ¿Qué es CVS?

CVS son las iniciales de Concurrent Version System. Es un tipo de software que se usa para seguir la pista de los cambios que se va realizando sobre un conjunto de ficheros (texto, graficos, etc). En definitiva, CVS ayuda a un grupo de gente trabajar sobre el mismo proyecto

2 ¿Qué no es CVS?

- No construye software. No es capaz de reconstruir un programa completo a partir de 'Makefiles' (o similares). Es tarea del grupo crear el 'Makefile' correspondiente para cada directorio.
- No sustituye a la cominicacion de los desarrolladores. Si bien CVS comunica de forma automatica que se realizan modificaciones sobre los ficheros y es capaz de determinar "algunos conflictos", CVS no esta diseñado para ningun lenguaje en particular por lo que no pude determinar los conflictos lógicos (de programacion) que pueden derivarse de la modificación simultanea sobre un grupo de ficheros.

3 ¿Por qué CVS?

Porque sigue la filosofia ampliamente extendida en la comunidad linux: todos pueden aportar algo sin modificar lo que otros han realizado.

Esto no signigica la necesidad de software UNIX. Tan solo hace falta las herramientas que utilice el grupo en el desarrollo y un cliente CVS. Los clientes graficos para Macinstosh, Unix y Windows 95/NT puede encontrarse en la dirección http://download.cyclic.com/pub. Los usuarios Linux encontrarán el paquete CVS libremente en su distribución (sin interface grafico).

En esta documentacion no se tratará NINGUN cliente gráfico ya que cada uno presenta su propio interface. Aqui se verá como obtener, modificar y actualizar los ficheros mediante lineas de comandos. Mas concretamente, se verá como hacerlo via Linux, ya que es el sistema más usado en el grupo.

4 Conceptos Básicos

CVS almacena todos los ficheros en un "Deposito Centralizado". El "depósito" pude contener directorios y ficheros en un árbol arbitrario.

Cada versión de un fichero tiene un único número de revisión de la forma '1.1', '1.2', '1.3.2.2' e incluso '1.3.2.2.4.5'.

La revisión '1.1' siempre será la primera, y a partir de ella se van generando las demas. Las versiones presentan una estructura del siguiente tipo.

```
                    /-- 1.2.2.3.2.1
                   /
```

```
          /-- 1.2.2.1 -- 1.2.2.2 -- 1.2.2.3 -- 1.2.2.4
         /
1.1 -- 1.2 -- 1.3 -- 1.4 -- .........
         \
 \--
```

En lo que sigue supondremos que las versiones son consecutivas:

```
1.1 -- 1.2 -- 1.3 -- 1.4 -- .........
```

La principal razón es que cuando se crea una nueva rama, resulta bastante complicado fusionar distintas revisiones de distintas ramas: Si un programador modifica la revisión 1.2.2.1 creará la revisión 1.2.2.2. Si otro programador modifica la revisión 1.2 creará la revisión 1.3. En consecuencia la fusión de las versiones 1.2.2.2 y 1.3. deberá hacerse totalmente manual, mientras que CVS puede realizar (con un poco de suerte) fusiones automáticas si las versiones fuesen consecutivas.

5 Primeros Pasos

Antes de empezar a trabajar, es necesario establecer algunas variables de entorno. Las mas importantes son:

CVSROOT Indica donde se encuentra el "depósito" y como se va a acceder a él. Nosotros usaremos solo una. La declaracion de la variable es como sigue:

```
export CVSROOT=:pserver:usuario\@leo.ugr.es:/home/gte/elvira/cvsroot
```

Donde usuario es: acu, smc, lci, jhg, ldaniel, jgamez, asc, elvira.

Los programadores deben de usar su popio login. Deben olvidarse de usar el login "elvira", ya que de usarse no se podra saber quien ha realizado las modificaciones sobre los ficheros.

CVSEDIT Indica el editor que se activará cuando se realice una modificacion.

Mas concretamente, cuando se realiza una modificación 'salta' un editor para que el programador introduzca las modificaciones que ha realizado en un fichero. El editor 'saltará' para cada fichero modificado, por lo que conviene usar el editor que a cada uno le resulte mas amigable. Yo personalmente recomiendo nedit ya que 'colorea' el documento según el tipo de texto que edita (C, Java, script, etc). El modo de declararla será:

```
export CVSEDIT=nedit
```

Por defecto, en Linux el editor es el VIM y en Win95/NT es el notepad.exe

El segundo paso consiste en crear un directorio local y cambiarse a él. Supondré que ese directorio se llama DESARROLLO, con lo que los pasos a realizar son:

```
mkdir DESARROLLO
cd DESARROLLO
```

MUY IMPORTANTE: **TODOS** los comandos que se ejecuten con CVS deberá realizarse a partir en este Directorio. La norma a seguir es:

1. Nos vamos al directorio `~/DESARROLLO`
2. Se ejecuta el comando CVS correspondiente en este directorio

6 Sesiones con CVS

6.1 Inicio de la Sesión

Para acceder al "depósito" de ficheros, una vez declaradas las variables e introducidos en el directorio `~/DESARROLLO`, ejecutaremos:

```
cvs login
```

Con lo que nos pedira el password de usuario. Si no se producen errores estamos en condiciones de seguir con los siguientes pasos.

Ejemplo.

```
ldaniel#~/DESARROLLO >cvs login
(Logging in to ldaniel@leo.ugr.es)
CVS password:
ldaniel#~/DESARROLLO >
```

6.2 Obteniendo una Versión: Comando checkout

Para obtener todos los ficheros java deberá ejecutarse

```
cvs checkout bayelvira
```

Ejemplo.

```
ldaniel#~/DESARROLLO >cvs checkout bayelvira
cvs server: Updating bayelvira
U bayelvira/ASCII_CharStream.java
U bayelvira/BayesNetConstants.java
U bayelvira/BayesNetParse.java
U bayelvira/BayesNetParseConstants.java
U bayelvira/BayesNetParseTokenManager.java
U bayelvira/BayesNetTokenManager.java
.......
U bayelvira/compiler.jj
ldaniel#~/DESARROLLO >
```

En el directorio DESARROLLO se habra creado un directorio llamado `bayelvira` con todos los ficheros java. Tambien se crea un directorio llamado *~/DESARROLLO/bayelvira/CVS*. NO DEBEN EDITARSE los ficheros que se encuentran en este directorio ya que CVS utiliza estos ficheros para controlar los accesos, modificaciones, etc que se realicen sobre los ficheros de bayelvira tanto local como remotos.

6.3 El directorio CVSROOT y otros

CUIDADO: Si realizamos el comando

```
ldaniel#~/DESARROLLO >cvs checkout
```

obtendremos todos los directorios que cuelgen de `/home/gte/elvira/cvsroot` (ver declaracion de la variable CVSROOT).

Esto significa que hay que ser meticulosos ya que la modificación de los ficheros del directorio `~/DESARROLLO/CVSROOT` puede afectar al funcionamiento de CVS. En el caso de que por algún error, nos trajesemos de leo.ugr.es algún directorio distinto de bayelvira, lo mejor que podemos hacer es borrarlo con el comando **rm** de Unix o semejante (en otros sistemas).

6.4 Editando y Compilando Ficheros Java

Esto lo hará cada uno según su costrumbre. Tan solo hay que tener presente algunas normas de conducta:

1. **No deben borrarse ficheros**. Aunque algunos ficheros dejen de usarse, conviene mantetener TODOS los ficheros salvo que todo el grupo, de mutuo acuerdo, considere la necesidad de su borrado. Ver *Borrando Ficheros* para saber como realizar esta operación.

2. **Si algún fichero depende de otro**, deben usarse trayectos relativos y nunca globales (ya que no tienen porqué coincidir los trayectos globales de las maquinas de todos los programadores).

3. **No actualizar el "depósito" si no se ha modificado un fichero por completo**. Es decir, si recogemos una versión de un fichero y lo modificamos, no debe de actualizarse el "depósito" hasta que no se esté seguro de que el fichero no contiene errores. Debe ser el programador que modifica el fichero el que debe depurar todos los posibles errores. Esta tarea no debe delegarse al resto del grupo.

6.5 Actualizando Ficheros Modificados: Comando commit

Una vez que hemos editado y modificado algunos ficheros, y se está seguro de que todo funciona, será necesario actualizar el "depósito". Esto se hace con el comando `cvs commit bayelvira`

Ejemplo.

Suponer que se modifican los ficheros *bayelvira/Token.java* y *bayelvira/Link.java* y queremos

actualizr el "depósito" con nuestra nuevas versiones. Veamos una sesion para actualizar estos nuevos ficheros simultaneamente (las lineas que empiezan con # son comentarios).

```
# Actualizamos los nuevos ficheros
ldaniel#~/DESARROLLO >cvs commit bayelvira

#En este momento nos saltará el editor que tengamos por defecto.
#Entonces introduciremos las modificaciones que hayamos realizado
#Por ejemplo, una sesión con el vim puede ser la siguiente:
CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS: ' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  bayelvira/Link.java bayelvira/Token.java
CVS: ----------------------------------------------------------------------

Link.java:  He añadido una nueva funcion cuyo objetivo es
             bla, bla, bla

Token.java: He añadido el Token B para que funciones mas optimo

# Las lineas que empiezan por CVS es lo que genera de forma automática
# CVS y no seran almacenadas en el "depósito"
# El resto del documento debe de introducirlo el programador
# Notar que los comentarios son comunes para LOS DOS ficheros modificados,
# por lo que conviene poner de forma explicita qué modificación se
# ha realizado sobre cada fichero. En nuestro caso, he puesto las
# modificaciones que se han realizado en Link.java y en Token.java
# Cuando salgamos del editor, el shell de comandos debe mostrar lo siguiente:

ldaniel#~/DESARROLLO >cvs commit bayelvira
Checking in bayelvira/Link.java;
/home/gte/elvira/cvsroot/bayelvira/Link.java,v  <--  Link.java
new revision: 1.3; previous revision: 1.2
done
Checking in bayelvira/Token.java;
/home/gte/elvira/cvsroot/bayelvira/Token.java,v  <--  Token.java
new revision: 1.10; previous revision: 1.9
done
```

La sesión anterior es un ejemplo de como introducir modificaciones sobre varios ficheros simultaneamente en UN solo 'golpe de comando'. Pero este modo de trabajar presenta 'un serio' inconveniente y es que CVS guarda para cada version de un fichero los comentarios

que introduce el programador. Esto quiere decir que en la nueva versión del fichero *Link.java* se almacenará el comentario que hemos introducido anteriormente, es decir:

```
Link.java:  He añadido una nueva funcion cuyo objetivo es
             bla, bla, bla

Token.java: He añadido el Token B para que funciones mas optimo
```

Pero, en la nueva versión del fichero *Token.java* también se almacenara el mismo comentario.

Como puede verse, esto no tiene sentido salvo que exista una relación muy muy directa entre ambos ficheros, en el sentido de que las modificaciones de uno puede repercutir en el otro y en consecuencia su modificación.

Lo normal es que cada nueva version de un fichero tenga su propio comentario. **Y esta es la norma que debe de adoptarse** (digamos que es una norma de conducta más). Para esto, habrá que actualizar el "depósito" haciendolo fichero a fichero. Para la situación anterior, deberiamos de haber actuado como sigue:

```
# EN PRIMER LUGAR Actualizamos el fichero Link.java
#En este momento nos saltará el editor que tengamos por defecto.
#Entonces comentaremos las modificaciones que hayamos realizado en Link.java
#Por ejemplo, una sesión con el vim puede ser la siguiente:


CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with 'CVS: ' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  bayelvira/Link.java
CVS: ----------------------------------------------------------------------
He añadido una nueva funcion cuyo objetivo es
             bla, bla, bla

# Notar que ahora los comentarios hacen referencia a bayelvira/Link.java,
# por lo que ahora no es necesario poner de forma explicita qué  modificación
# se ha realizado sobre el fichero Link.java.
# Cuando salgamos del editor, el shell de comandos debe mostrar lo siguiente:

ldaniel#~/DESARROLLO >cvs commit bayelvira/Link.java
Checking in bayelvira/Link.java;
/home/gte/elvira/cvsroot/bayelvira/Link.java,v  <--  Link.java
new revision: 1.4; previous revision: 1.3
done
```

```
# ================================================
# EN SEGUNDO LUGAR Actualizamos el fichero Token.java

ldaniel#~/DESARROLLO >cvs commit bayelvira/Token.java

#En este momento nos saltará el editor que tengamos por defecto.
#Entonces comentaremos las modificaciones que hayamos realizado en Token.java
#Por ejemplo, una sesión con el vim puede ser la siguiente:
CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS: ' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  bayelvira/Token.java
CVS: ----------------------------------------------------------------------
He añadido el Token B para que funciones mas optimo

# Notar que ahora los comentarios hacen referencia a bayelvira/Token.java,
# por lo que ahora no es necesario poner de forma explicita qué modificación
# se ha realizado sobre el fichero Token.java.
# Cuando salgamos del editor, el shell de comandos debe mostrar lo siguiente:

ldaniel#~/DESARROLLO >cvs commit bayelvira/Token.java
Checking in bayelvira/Token.java;
/home/gte/elvira/cvsroot/bayelvira/Token.java,v  <--  Token.java
new revision: 1.11; previous revision: 1.10
done
```

Atención: Puede ser que al crear nuevas versiones, CVS no nos deje hacerlo. Ver Comando UPDATE.

6.5.1 Notificación Automática de Revisiones

Cada vez que un programador somete una nueva versión mediante el comando **commit**, automáticamente se lanza un e-mail a todos los programadores. El e-mail presenta el siguiente aspecto:

```
Subject:[ELVIRA] Nueva Modificacion
   Date:Mon, 27 Jul 1998 18:05:08 +0100
  From:Luis Daniel Hernandez Molinero <ldaniel@leo.ugr.es>
    To:proyecto@leo.ugr.es
```

```
Update of /home/gte/elvira/cvsroot/bayelvira
In directory leo.ugr.es:/tmp/cvs-serv9161/bayelvira

Modified Files:
        Token.java
Log Message:
He añadido el Token B para que funcione mas optimo
```

Su significado es el siguiente:

El programador *ldaniel* ha realizado una modificación y se notifica al grupo *proyecto@leo.ugr.es*. La modificación se ha realizado en el directorio `/home/gte/elvira/cvsroot/bayelvira` sobre el fichero `Token.java`. El mensaje que ha introducido el programador cuando le 'ha saltado' el editor es: `He añadido el Token B para que funcione mas optimo`

Notar la importancia de introducir **BUENOS** comentarios al ejecutar el comando **commit**. Además dichos mensajes pueden visualizarse via WEB por lo que un buen comentario puede ayudarnos a seguir el desarrollo de los ficheros.

6.6 Añadiendo Nuevos Ficheros: Comando add

Para añadir ficheros nuevos a una versión los pasos a seguir son:

1. Creamos un nuevo fichero en `~/DESARROLLO/bayelvira`. Lo editamos, compilamos, etc ... Y cuando este listo, pasamos a los siguientes pasos.
2. Le decimos al servidor CVS que se ha creado un nuevo fichero con el comando `cvs checkout bayelvi`
3. Nos vamos al directorio bayelvira `cd bayelvira`.
4. Ejecutamos el comando `cvs add nuevo.java`
5. Ejecutamos el comando `cvs commit nuevo.java`

Ejemplo.

Suponer que se crea el fichero *nuevo.java* en el directorio *bayelvira* y queremos añadirlo al "depósito". Veamos una sesion para añadir este nuevo fichero (las lineas que empiezan con # son comentarios).

```
# CREAMOS EL NUEVO FICHERO
ldaniel#~/DESARROLLO >nedit bayelvira/nuevo.java

# INTENTAMOS AÑADIR EL FICHERO
ldaniel#~/DESARROLLO >cvs add bayelvira/nuevo.java
cvs add: in directory .:
cvs [add aborted]: there is no version here; do 'cvs checkout' first

#Notar que es necesario decirle antes al servidor CVS que "registre"
```

```
# el nuevo fichero. Esto se hace con el siguiente comando.

ldaniel#~/DESARROLLO >cvs checkout bayelvira
? bayelvira/nuevo.java
cvs server: Updating bayelvira

# Ahora sí estamos en condiciones de añadir el nuevo fichero.
# Notar que si se hace en el directorio ~/DESARROLLO el comando
# no funciona como se muestra en el siguiente ejemplo

ldaniel#~/DESARROLLO >cvs add bayelvira/nuevo.java
cvs add: in directory .:
cvs [add aborted]: there is no version here; do 'cvs checkout' first

# Es decir, es necesario 'desplazarse' al directorio bayelvira

ldaniel#~/DESARROLLO >cd bayelvira
ldaniel#~/DESARROLLO/bayelvira >cvs add nuevo.java
cvs server: scheduling file 'nuevo.java' for addition
cvs server: use 'cvs commit' to add this file permanently

# Por ultimo, añadimos el fichero al "depósito"
ldaniel#~/DESARROLLO/bayelvira >cvs commit nuevo.java
RCS file: /home/gte/elvira/cvsroot/bayelvira/nuevo.java,v
done
Checking in nuevo.java;
/home/gte/elvira/cvsroot/bayelvira/nuevo.java,v  <--  nuevo.java
initial revision: 1.1
done
```

6.7 Borrando Ficheros: Comando remove

Para borrar ficheros los pasos a seguir son:

1. **MANDAR UN e-mail A TODOS LOS PROGRAMADORES** para ver la necesidad de borrar ese fichero, y si todos están de acuerdo, pasamos a los siguientes pasos.

2. Borrar el fichero de `~/DESARROLLO/bayelvira/viejo.java`.

3. Ejecutar el comando `cvs remove bayelvira/viejo.java`

4. Ejecutar el comando `cvs commit bayelvira/viejo.java`

Ejemplo.
Suponer que se borra el fichero *viejo.java* en el directorio *bayelvira* (las lineas que empiezan con # son comentarios).

```
# En primer lugar borramos el fichero en nuestro directorio local

ldaniel#~/DESARROLLO >rm bayelvira/viejo.java

# En segundo lugar, el decimos al servidor que 'registre' su borrado.

ldaniel#~/DESARROLLO >cvs remove bayelvira/viejo.java
cvs server: scheduling `bayelvira/viejo.java' for removal
cvs server: use 'cvs commit' to remove this file permanently

# En tercer lugar, actualizamos el depósito ejecutando:

ldaniel#~/DESARROLLO >cvs commit bayelvira/viejo.java

# Como siempre que ejecutemos el comando commit, nos 'saltara' el
# editor que tengamos por defecto. Un ejemplo de una sesión con VIM es:

CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS: ' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Removed Files:
CVS:  bayelvira/viejo.java
CVS: ----------------------------------------------------------------------
Se ha borrado el fichero viejo.java.
Las funciones de este fichero se han puesto en los
ficheros file1.java, file2.java, etc .....

# Al salir del editor, nos encontraremos el shell de comandos con
# las siguientes lineas:

ldaniel#~/DESARROLLO >cvs commit bayelvira/viejo.java
Removing bayelvira/viejo.java;
/home/gte/elvira/cvsroot/bayelvira/viejo.java,v  <--  viejo.java
new revision: delete; previous revision: 1.1
done
```

MUY IMPORTANTE **Una vez que se ejecute el comando 'commit' el fichero dejará de existir por lo que se insiste en la necesidad de mandar al grupo un e-mail de que dicha operacion va a realizarse**

6.8 Fusión automática de Ficheros: Comando update

Imaginemos que un fichero ha sido recuperado del "depósito" por dos programadores y que cada uno ha realizado una serie de modificaciones. ¿Qué ocurre en la actualización? Lo mejor es verlo con un ejemplo:

Supongamos que existe el fichero *Numeros.java* (versión 1.5) con el siguiente contenido:

```
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
```

Y supongamos que un programador modifica su contenido a:

```
001
002
003
004
005
006
007
008
009
010
xxx1  // Nueva linea Introducida por Programador
011
012
013
014
015
```

y a continuación actualiza el "depósito" antes que el otro con el comando "commit" generando la versión 1.6. Es decir, El primer programador ejecuta:

```
[Programador@alhambra ~/DESARROLLO]$ cvs commit bayelvira/Numeros.java
Checking in bayelvira/Numeros.java;
```

```
/home/gte/elvira/cvsroot/bayelvira/Numeros.java,v  <--  Numeros.java
new revision: 1.6; previous revision: 1.5
done
```

Supongamos que ahora el segundo programador, al que llamaremos *ldaniel* realiza la siguiente modificación:

```
001
002
003
004
005
yyy1 // Nueva linea introducida por ldaniel
006
007
008
009
010
011
012
013
014
015
```

Cuando este ultimo programador vaya a actualizar el "depósito" con el comando "commit" se encontrará con el siguiente mensaje:

```
ldaniel#~/DESARROLLO > cvs commit bayelvira/Numeros.java
cvs server: Up-to-date check failed for ‘bayelvira/Numeros.java’
cvs [server aborted]: correct above errors first!
```

Este mensaje de error significa que la versión de *ldaniel* (que es una versión corregida de la 1.5) es posterior a la que ya ha introducido el primer programador en el "depósito" (que es la versión 1.6).

Para corregir los errores será necesario ejecutar el comando **update** con objeto de que CVS **fusione** la versión de *ldaniel* con la 1.6.

```
ldaniel#~/DESARROLLO >cvs update bayelvira/Numeros.java
RCS file: /home/gte/elvira/cvsroot/bayelvira/Numeros.java,v
retrieving revision 1.5
retrieving revision 1.6
Merging differences between 1.5 and 1.6 into Numeros.java
M bayelvira/nuevo2.java
```

El resultado de la fusión de versiones es la siguiente:

```
001
002
003
004
005
yyy1 // Nueva linea introducida por ldaniel
006
007
008
009
010
xxx1 // Nueva linea Introducida por Programador
011
012
013
014
015
```

En este caso la **fusión** ha sido realizada de forma **automática**. Sin embargo, esto no es lo usual. Lo más común es que se produzcan "conflictos" entre las versiones en cuyo caso será necesario realizar la **fusión manual**. Para esta situación ver Resolución de Conflictos.

6.9 Estado de los Ficheros: Comando status

El comando **status** informa sobre el estado de los ficheros de un "depósito". Su sintaxis puede ser:

```
cvs status
cvs status directorio
cvs status directorio/fichero
```

Ejemplo

```
ldaniel#~/DESARROLLO >cvs status
===================================================================
File: Token.java        Status: Up-to-date

   Working revision:    1.11
   Repository revision: 1.11    /home/gte/elvira/cvsroot/bayelvira/Token.java,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: TokenMgrError.java        Status: Up-to-date

   Working revision:    1.1
```

```
   Repository revision: 1.1     /home/gte/elvira/cvsroot/bayelvira/TokenMgrError
.java,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: compiler.jj       Status: Up-to-date

   Working revision:    1.1
   Repository revision: 1.1     /home/gte/elvira/cvsroot/bayelvira/compiler.jj,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: nuevo.java        Status: Up-to-date

   Working revision:    1.5
   Repository revision: 1.5     /home/gte/elvira/cvsroot/bayelvira/nuevo.java,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: nuevo2.java       Status: Locally Modified

   Working revision:    1.6
   Repository revision: 1.6     /home/gte/elvira/cvsroot/bayelvira/nuevo2.java,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Los posibles estados de un ficheros son

Up-to-date El fichero es identico con la última revisión del "depósito" de la rama en uso.

Locally Modified Has editado el fichero, y no se han 'committed' los cambios.

Locally Added Has añadido un fichero con **add**, y no se han 'committed' los cambios.

Locally Removed Has borrado un fichero con **remove**, y no se han 'committed' los cambios.

Needs Checkout Alguien a creado una nueva revisión del fichero en el "depósito". Lo normal es usar **update** en vez de **checkout** para obtener la última revisión.

Needs Merge Alguien a creado una nueva revisión del fichero en el "depósito" y tu también has hecho modificaciones en el fichero.

Unresolved Conflict Es como Locally Modified, excepto que un comando previos **update** dio un conflicto. Es necesario resolver el conflicto.

Unknown CVS no sabe nada sobre este fichero. Por ejemplo, haber creado un nuevo fichero y no haber realizado **add**.

6.10 Cerrando la Sesión y Borrando Directorio Local de Trabajo: Comando release

Una vez realizadas las modificaciones oportunas en los ficheros, y si estamos en condiciones de **Cerrar la Sesión** (los ficheros no presentan errores) podemos comprobar que podemos irnos a casa sin problemas posteriores ejecutando el comando **release**. Este comando comprueba que nuestros ficheros están en orden con respecto al "depósito".

Ejemplo

```
ldaniel#~/DESARROLLO >cvs release bayelvira
M Prueba.java
M nuevo2.java
You have [2] altered files in this repository.
Are you sure you want to release module ‘bayelvira’: n
** ‘release’ aborted by user choice.
```

En este caso se ha querido hacer una una nueva versión de *bayelvira* 'a un solo golpe de comando'. En este caso CVS ha detectado que los ficheros *Prueba.java* y *nuevo2.java* han sido modificados (por algún comando anteriormente ejecutado –p.e. update, checkout– o porque simplemente lo hemos editado nosotros). **EN ESTOS CASOS SIEMPRE** hay que responder a la pregunta `Are you sure you want to release module ‘bayelvira’:` con un NO ('n'), y ejecutar los cambios oportunos. NO responder nunca con un YES.

NOTA: Este comando no siempre es necesario ejecutarlo, pero si conveniente. La ventaja de ejecutar **release** es que 'a un solo golpe de comando' nos dice el estado de nuestros ficheros con respecto al "depósito".

Es por ésto que es un comando que también suele utilizarse para conocer el estado actual de nuestra versión local antes de empezar a trabajar en una nueva sesión.

El uso más amplio del comando **release** es para LIMPIAR nuestro directorio de trabajo. Si uno decide borrar una copia de trabajo de *bayelvira*, un modo de hacerlo es:

```
ldaniel#~/DESARROLLO > cd ~/DESARROLLO
ldaniel#~/DESARROLLO > rm -r bayelvira
```

Pero es mejor ejecutar el comando release del siguiente modo:

```
ldaniel#~/DESARROLLO >cvs release -d bayelvira/
```

Si no sale ningún mensaje de error, significa que nuestro trabajo está actualizado con respecto al "depósito" y por lo tanto el borrado de `bayelvira` no es importante. Sin embargo, si nos saliese un mensaje como el anterior y decimos YES a la pregunta:

```
ldaniel#~/DESARROLLO >cvs release -d bayelvira
M Prueba.java
M nuevo2.java
You have [2] altered files in this repository.
Are you sure you want to release (and delete) module ‘bayelvira’: y
```

Nuestro directorio local será borrado y por tanto los cambios que se hayan realizado en *Prueba.java* y *nuevo2.java* se perderán.

NOTA: Sólo debe de responderse YES cuando salga el siguiente mensaje:

```
ldaniel#~/DESARROLLO >cvs release -d bayelvira
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module ‘bayelvira’: y
```

7 Resolución de Conflictos: Comando update

7.1 Ejemplo 1

Imaginemos que un fichero ha sido recuperado del "depósito" por dos programadores y que cada uno ha realizado una serie de modificaciones.

Supongamos que existe el fichero *Numeros.java* (versión 1.9) con el siguiente contenido:

```
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
```

Y supongamos que un programador modifica su contenido a:

```
001
002
```

```
003
004
005
xxx1 // Linea introducida por Programador
xxx2
xxx3
006
007
008
009
010
011
012
013
014
015
```

y a continuación actualiza el "depósito" antes que el otro con el comando "commit" generando la versión 1.10. Es decir, El primer programador ejecuta:

```
[Programador@alhambra ~/DESARROLLO]$ cvs commit bayelvira/Numeros.java
Checking in bayelvira/Numeros.java;
/home/gte/elvira/cvsroot/bayelvira/Numeros.java,v  <--  Numeros.java
new revision: 1.10; previous revision: 1.9
done
```

Supongamos que ahora el segundo programador, al que llamaremos *ldaniel* realiza la siguiente modificación:

```
001
002
003
004
005
006
007
008
009
010  // Se han borrado las lineas 011 - 014  por ldaniel
015
```

Cuando este ultimo programador vaya a actualizar el "depósito" con el comando "commit" se encontrará con el siguiente mensaje:

```
ldaniel#~/DESARROLLO > cvs commit bayelvira/Numeros.java
cvs server: Up-to-date check failed for ‘bayelvira/Numeros.java’
cvs [server aborted]: correct above errors first!
```

Este mensaje de error significa que la versión de *ldaniel* (que es una versión corregida de la 1.9) es posterior a la que ya ha introducido el primer programador en el "depósito" (que es la versión 1.10).

Para corregir los errores será necesario ejecutar el comando **update** con objeto de que CVS **fusione** la versión de *ldaniel* con la 1.10.

```
ldaniel#~/DESARROLLO >cvs update bayelvira/Numeros.java
RCS file: /home/gte/elvira/cvsroot/bayelvira/Numeros.java,v
retrieving revision 1.9
retrieving revision 1.10
Merging differences between 1.9 and 1.10 into Numeros.java
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in bayelvira/nuevo2.java
C bayelvira/Numeros.java
```

En este caso CVS es incapaz de realizar la **fusión automática** de documentos. Por lo que será necesario realizar una **fusión manual** para resolver el conflicto mediante la edición del fichero.

El resultado de la **semi-fusión** de versiones es la siguiente:

```
001
002
003
004
005
<<<<<<< Numeros.java
=======
xxx1 // Linea introducida por Programador
xxx2
xxx3
>>>>>>> 1.10
006
007
008
009
010  // Se han borrado las lineas 011 - 014  por ldaniel
015
```

El modo de entender la **semi-fusión** es verlo de la siguiente manera: La versión de *ldaniel* no contiene algunas lineas de la versión 1.10. CVS entonces solapa al versión 1.10 sobre la versión de *ldaniel*

La parte que se solapa entre las versiones son las marcadas por: `<<<<<<<`, `=======` y `>>>>>>>`. Más concretamente:

¡¡¡¡¡¡¡ Numeros.java Indica que aquí comienza el solapamiento.

¿¿¿¿¿¿¿ **1.10** Indica que ahí se acaba el solapamiento de la versión 1.10 sobre el fichero de *ldaniel*

En este caso el borrado de las marcas ¡¡¡¡¡¡¡, `=======` y `>>>>>>>` será suficiente para obtener una versión compatible con las versiones 1.10 y la de *ldaniel*.

Para el significado de `=======` pasar al siguiente ejemplo.

7.2 Ejemplo 2

Supogamos que dos programadores recuperan el fichero *Numeros.java* (versión 1.9) con el siguiente contenido:

```
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
```

Supongamos que el primer programador crea la versión 1.10 con el siguiente fichero mediante el comando **commit**:

```
001
002
003
004
005
xxx1 // Linea introducida por Programador
xxx2
xxx3
006
007
008
009
010
011
```

```
012
013
014
015
```

Y que *ldaniel* sobre la versión 1.9 crea el siguiente fichero.

```
001
002
003
004
005
yyy1 // Estas son las lineas introducidas por ldaniel
yyy2
yyy3
006
007
008
009
010  // Se han borrado las lineas 011 - 014  por ldaniel
015
```

Observar que en este caso *ldaniel* ha introducido lineas en el mismo lugar que donde lo hizo el primer programador. Tras ejecutar el comando **commit**, *ldaniel* recibira el mensaje de error

```
ldaniel#~/DESARROLLO >cvs commit bayelvira/Numeros.java
cvs server: Up-to-date check failed for ‘bayelvira/Numeros.java’
cvs [server aborted]: correct above errors first!
```

Por lo que *ldaniel* tendrá que ejecutar el comando **update** y vemos que se producen conflictos

```
ldaniel#~/DESARROLLO >cvs update bayelvira/Numeros.java
RCS file: /home/gte/elvira/cvsroot/bayelvira/Numeros.java,v
retrieving revision 1.9
retrieving revision 1.10
Merging differences between 1.9 and 1.10 into Numeros.java
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in bayelvira/Numeros.java
C bayelvira/Numeros.java
```

Si editamos el nuevo fichero, vemos el siguiente solapamiento:

```
001
002
003
```

```
004
005
<<<<<<< Numeros.java
yyy1 // Estas son las lineas introducidas por ldaniel
yyy2
yyy3
=======
xxx1 // Linea introducida por Programador
xxx2
xxx3
>>>>>>> 1.10
006
007
008
009
010  // Se han borrado las lineas 011 - 014  por ldaniel
015
```

La parte que se solapa entre las versiones son las marcadas por: `<<<<<<<`, `=======` y `>>>>>>>`. Más concretamente:

¡¡¡¡¡¡¡ **Numeros.java** Indica que aquí comienza el solapamiento.

¿¿¿¿¿¿¿ **1.10** Indica que ahí se acaba el solapamiento de la versión 1.10 sobre el fichero de *ldaniel*

======= Indica la separación de lo que ha introducido el primer programador y *ldaniel*. Más concretamente.

- El código comprendido entre `<<<<<<< Numeros.java` y `=======` es el codigo introducido por el último programador (en este caso *ldaniel*).
- El código comprendido entre `=======` y `>>>>>>> 1.10` es el codigo introducido por el primer programador.

Quizás ahora el ejemplo 1 le ayude a entender mejor el significado de `=======`

Será ahora tarea del **ULTIMO** programador (*ldaniel*) **RESPETAR AL MÁXIMO** el trabajo realizado por el **PRIMER** programador. Es decir, *ldaniel* deberá **ADAPTAR** su código al código del primer programador.

Por ejemplo un fichero resultante puede ser:

```
001
002
003
004
005
```

```
//  Aqui comenzaba el solapamiento de la versión 1.10 de Numeros.java
if (TRUE)
{
   yyy1 // Estas son las lineas introducidas por ldaniel
   yyy2
   yyy3
} else
  {
xxx1 // Linea introducida por Programador
xxx2
xxx3
}
006
007
008
009
010  // Se han borrado las lineas 011 - 014  por ldaniel
015
```

Posteriormente el ultimo programador generará la versión 1.11:

```
ldaniel#~/DESARROLLO >cvs commit  bayelvira/Numeros.java
Checking in bayelvira/Numeros.java;
/home/gte/elvira/cvsroot/bayelvira/Numeros.java,v  <--  Numeros.java
new revision: 1.11; previous revision: 1.10
done
```

7.3 ¿Qué hacer en caso de Conflicto?

Por favor, antes de leer esta sección, lea los ejemplos anteriores.

Como se ha visto en los dos ejemplos anteriores, al ejecutar el comando **update**, CVS MACHACA nuestra versión solapando la versión que se encuentra en el depósito. En consecuencia, nuestra versión queda modificada.

Si no queremos perder nuestra versión (p.e. aún no se entiende muy bien lo del solapamiento o simplemente no queremos perderla), un modo de actuar sería la siguiente sesión (las lineas que empiezan por # son comentarios):

```
# Copiar nuestro version de fichero con otro nombre
ldaniel#~/DESARROLLO > cp bayelvira/Numeros.java bayelvira/MiVersionNumeros.java

# Solapar la versión del "depósito" con nuestra versión

ldaniel#~/DESARROLLO > cvs update bayelvira/Numeros.java
RCS file: /home/gte/elvira/cvsroot/bayelvira/Numeros.java,v
retrieving revision 1.9
```

```
retrieving revision 1.10
Merging differences between 1.9 and 1.10 into Numeros.java
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in bayelvira/Numeros.java
C bayelvira/Numeros.java

# Editar los dos ficheros
ldaniel#~/DESARROLLO > nedit bayelvira/MiVersionNumeros.java &
ldaniel#~/DESARROLLO > nedit bayelvira/Numeros.java &

# Realizar los cambios oportunos en Numeros.java y salvarlo.
# Someter la nueva versión.

ldaniel#~/DESARROLLO >cvs commit  bayelvira/Numeros.java
Checking in bayelvira/Numeros.java;
/home/gte/elvira/cvsroot/bayelvira/Numeros.java,v  <--  Numeros.java
new revision: 1.11; previous revision: 1.10
done

# BORRAR el antiguo fichero
ldaniel#~/DESARROLLO > rm bayelvira/MiVersionNumeros.java
```

NOTA: Es importante borrar el antiguo fichero ya que CVS puede producir mensajes innecesarios en un futuro. Por ejemplo, si no borrasemos el fichero y el que ha generado la última versión (la 1.11) ejecuta **release** obtendra el siguiente mensaje:

```
ldaniel#~/DESARROLLO >cvs release bayelvira
? MiVersionNumeros.java
You have [0] altered files in this repository.
Are you sure you want to release module ‘bayelvira’: y
```

Indicando que CVS no sabe (?) de donde ha salido el fichero *MiVersionNumeros.java*

Por ultimo indicar que con objeto de que los solapamientos no nos traigan de cabeza, es MUY MUY conveniente documentar lo MÁXIMO posible toda edición que se realice sobre los ficheros.

8 Visualizando BAYELVIRA por la WEB

Los proyectos mediante CVS pueden visualizarse via WEB conectandonos a la dirección http://leo.ugr.es/cgi-bin/cvsweb

Tan sólo una observación en el uso de este interface: **No debe recuperarse versiones de los ficheros mediante la opción** `Save as...` **de los navegadores** ya que entonces

CVS no tendrá constancia de los ficheros que serán modificados. **Usar siempre el comando checkout o update**.

9 Quiero saber más sobre CVS

- http://www.loria.fr/~molli/cvs-index.html
- ftp://download.cyclic.com/pub/packages/RPMS/i386/
- http://www.cyclic.com/cvs/info.html