

PROCESO DE INTEGRACIÓN CON SCM

A lo largo de esta serie de artículos se ha intentado iniciar al lector en el uso y funcionamiento de las herramientas de control de versiones. En las primeras entregas se dio una introducción a este tipo de herramientas, se explicaron algunos de los términos más importantes, se proporcionó una guía de iniciación y se explicaron algunas de las principales metodologías empleadas en el uso de un SCM. En esta última entrega nos centraremos en el proceso más crítico en el manejo de un SCM: las integraciones, así como la operación más importante del mismo: el merge.

¿Qué es una integración?

Una integración es el proceso mediante el cual un cambio realizado por un desarrollador pasa a formar parte del producto.

Tal y como hemos venido hablando, cuando un desarrollador realiza algún cambio lo hace de forma local, en su propio espacio de trabajo, por lo que no formará parte del producto hasta que estos cambios sean introducidos en la línea principal de desarrollo.

La forma y el proceso de introducirlo vienen determinados en gran medida por la metodología de trabajo empleada. Pero salvo que el desarrollo de todo el producto lo lleve a cabo un único desarrollador siempre hay un punto en común y es la necesidad de combinar los cambios introducidos por los diferentes desarrolladores. Para poder combinar dichos cambios se emplea el merge.

Merge

Si solamente una persona ha modificado un elemento, está claro que para introducir este cambio es suficiente con sustituir la antigua versión con la nueva versión del mismo. Pero, ¿qué pasa cuando más de una persona ha modificado el mismo elemento? La primera persona sustituirá la antigua versión con la suya

nueva, pero si la segunda hace lo mismo, los cambios de la primera se perderían. Los controles de versiones permiten controlar estas situaciones y, gracias a ello, evitar la pérdida de cambios.

En la situación anterior, la segunda persona tendría que hacer un merge entre sus cambios y los de la primera persona, de forma que ninguno de ellos se perdiese. La operación de merge será la encargada de combinar los cambios de ambas personas de forma automática y, en algunos casos, con la intervención del usuario. Los cambios que la herramienta de merge puede mezclar de manera automática, se conocen como conflictos automáticos (o cambios), y, los cambios para los que necesita la intervención del usuario, se conocen como conflictos no automáticos. Estas herramientas pueden formar parte del control de versiones o ser externas al mismo, incluso algunos controles de versiones permiten trabajar con más de una herramienta de merge.

Para llevar a cabo un merge casi todas las herramientas de merge utilizan el merge de tres vías (three-way merge). En este tipo de merge participan tres versiones del elemento: la versión actual en el destino del merge (la versión actual del usuario), la versión origen del merge (de la cual queremos incorporar los cambios) y un antecesor común a ambas (el más cercano). A los dos primeros se les llama contribuidores y el antecesor común es la base.

El antecesor común es proporcionado por parte del control de versiones a la herramienta de merge. Se busca entre todas las versiones pertenecientes al historial del fichero que se desea mezclar, siendo la versión más cercana común, en la evolución de ambos contribuidores. Este cálculo, que puede complicarse sobremanera, es muy importante dentro del merge, dado que la complejidad del mismo vendrá determinada de lo bien o mal que se haya elegido este elemento (cuanto más próximo esté a los contribuidores más fácil será el merge). Para una búsqueda óptima del mismo, no sólo se empleará el parentesco entre las revisiones, sino que también han de utilizarse los links de merge existentes (esto último sólo pueden hacerlo aquellos controles de versiones que dispongan de este tipo de enlaces).

Para encontrar los conflictos se buscan los cambios que existen entre cada uno de los contribuidores con la base, para posteriormente compararlos entre sí y ver si pueden ser o no automatizados. La forma de automatizar los conflictos viene dada por las heurísticas (reglas) de merge, éstas son particulares para cada control de versiones pero la inmensa mayoría suelen coincidir. En las tablas que expondremos con heurísticas, se mostrarán las de la herramienta de "Plastic SCM" (una simplificación de las mismas).

El funcionamiento del merge varía en función del elemento sobre el que se realiza el merge. Por ello, para tratar de explicar mejor su funcionamiento estudiaremos cada uno de los tipos de forma independiente.

Merge de ficheros de texto

Es el más empleado de todos, dado que se emplea para mezclar ficheros de texto plano (texto sin formato). Este tipo de ficheros son los más frecuentes dentro del desarrollo de software y del uso de este tipo de herramientas en general.

Cuando un fichero es modificado de dos maneras distintas y se quieren unificar esas dos versiones en una sola, es necesario tratar de combinar esos cambios. Si los cambios se realizan en distintas partes del fichero, por ejemplo una versión ha modificado el principio del fichero y otra el final del mismo, la versión resultante del merge tendría el principio del fichero de la primera versión y el final del fichero de la segunda versión. Pero, ¿qué pasa cuando los cambios están desperdigados por el fichero en ambas versiones? En ese caso no es tan fácil verlo. Para ello la herramienta de merge busca todos los cambios realizados por cada uno de los contribuidores sobre la base. Estos cambios suelen buscarse a nivel de línea. Una vez haya encontrado todos estos cambios, que pueden ser que una línea haya sido borrada, añadida o modificada, tratará de automatizarlos en función a la heurística de merge mostrada en la Tabla 1:

Base	Contribuidor origen	Contribuidor destino	Resultado	Tipo de conflicto
A	A	A	A	Automático
A	A			Automático
A		A		Automático
A				Automático
A		B	?	No automático
A	B		?	No automático
A	A	B	B	Automático
A	B	A	B	Automático
A	B	B	B	Automático
A	B	C	?	No automático
	A		A	Automático
		A	A	Automático
	A	A	A	Automático
	A	B	?	No automático

Tabla 1. Heurística simplificada para el merge de ficheros de texto. Mostrándose para cada uno de los casos el tipo el resultado generado y el tipo de conflicto que es.

En general, cuando una línea sólo ha sido modificada por un contribuidor, esta línea se dejará con esa modificación en la versión resultante tal y como se hacía en el ejemplo cuando la región inicial del fichero había sido modificado por un solo contribuidor.

Como puede observarse, cuanto más cercana sea la base a los contribuidores del merge, menos cambios tendrán éstos sobre la misma y, por tanto, menos posibilidades de que se produzca un conflicto no automático.

La mayoría de las herramientas de merge mostrarán una interfaz para que el usuario resuelva los conflictos no automáticos y supervise los conflictos manuales en caso de que así lo desee. Estas interfaces muestran los 4 ficheros involucrados en el merge, que son: los dos contribuidores, la base y el fichero con el resultado del merge. Todas ellas permiten movernos por los distintos ficheros manteniendo de una u otra manera la correspondencia de las líneas que no han sido modificadas. De igual manera, permiten navegar entre los cambios y conflictos encontrados, así como modificar el resultado de los mismos, cambiando el contribuidor (o contribuidores) con el (los) que deseamos quedarnos, e incluso alterar el fichero de resultados editándolo manualmente. La forma de hacerlo es bastante diferente de unas herramientas a otras, podemos ver como sería en el caso de Mergetool la herramienta de merge de Plastic SCM en la Figura 1.

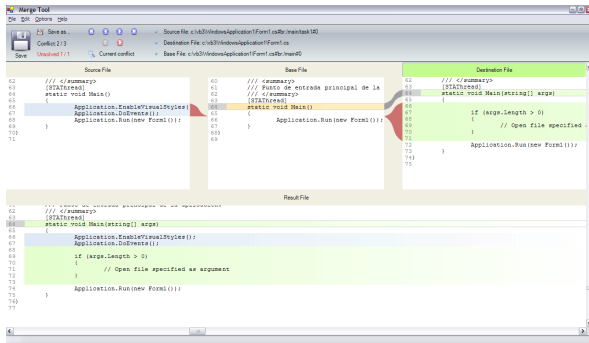


Figura 1. Captura de la herramienta para merge de ficheros de texto Mergetool

Merge de directorios

Existen controles de versiones que son capaces de versionar no sólo los ficheros, sino también los directorios.

Al igual que se modifica un fichero a lo largo del desarrollo de un producto, no es extraño que la estructura del directorio también cambie, moviendo los elementos de sitio, agrupándolos en carpetas, renombrándolos, etc. Son este tipo de cambios los que se gestionan mediante el versionado de directorios.

Un directorio puede sufrir diferentes cambios, sus elementos pueden borrarse o renombrarse, o pueden añadirse nuevos elementos.

A diferencia del merge de ficheros, en el caso del merge de directorios tenemos dos tipos de conflictos diferentes. Al igual que sucedía cuando en un fichero dos cambios afectaban al mismo fragmento que se solía producir un conflicto no automático, cuando dos cambios afectan al mismo elemento se suele producir un conflicto no automático. Un ejemplo de este caso es: cuando dos usuarios renombran el mismo elemento con dos nombres distintos, cuando se haga el merge será necesario elegir con cuál de los dos nombres se queda (o incluso darle un tercer nombre).

Pero en el caso de los directorios existe otro tipo de conflictos no automáticos, éstos son los conflictos de nombre. En un mismo directorio dos elementos diferentes no puede tener el mismo nombre, al no soportarlo los S.O. (Sistemas Operativos).

Un ejemplo de este caso es: si dos elementos distintos son renombrados con el mismo nombre, en el merge será necesario decidir para alguno de los dos elementos renombrados un nuevo nombre.

En la tabla 2 podemos ver la heurística para el merge de directorios de Plastic SCM, mostrando los distintos tipos de conflictos, la acción a tomar y su clasificación como conflicto automático o no automático:

Base	Contribuidor origen	Contribuidor destino	Resultado	Tipo de conflicto
file1	file1 -> file2	file1 -> file3	Un fichero renombrado con 2 nombres distintos -> elegir uno de los nombres y un nombre nuevo para el fichero	No automático
file1	file1	file1 -> file3	file3	Automático
file1	file1 -> file2	file1	file2	Automático
file1	file1 -> file3	file1 -> file3	?	No automático
file1	file1 -> file3	file1 -> file2	Los ficheros renombrados al mismo nombre -> renombrar uno de los ficheros a otro nombre	No automático
file1	file1 -> file3	file1 -> file2	file1 -> file2	Automático
file1	file1	file1 -> file3	file1 -> file3	Automático
			?	No automático
	file2	file2	Los ficheros con el mismo nombre -> Cambiar el nombre a uno o a los dos	No automático
	file2		file2	Automático
		file2	file2	Automático
			?	No automático
file1	file1 -> file2	file1 -> file2	Un fichero renombrado con el mismo nombre -> Cambiar el nombre a uno de ellos o a los dos	No automático
file1	file1	file1	file1 -> file2	Automático
file1	file1 -> file2	file1	file2	Automático

Tabla 2. Heurística simplificada para el merge de directorios. Mostrándose para cada uno de los casos el tipo el resultado generado, la acción a realizar y el tipo de conflicto que es.

Cuando el control de versiones no es capaz de gestionar los directorios, suelen perder la historia de aquellos elementos que han sido renombrados o movidos, quedando dividida la historia en lo que sucedió antes y después de ese cambio.

En el caso del merge de directorios suelen venir soportados por el propio control de versiones y no por herramientas externas. Esto es así porque una herramienta externa no tiene la posibilidad de identificar que un elemento es el mismo si ha cambiado de nombre al no disponer de información adicional.

A diferencia del caso anterior, en el merge de directorios la representación no es muy parecida entre los distintos controles de versiones (si es que lo soportan). En el caso de Plastic SCM, cuando un merge de directorios no es automático el usuario es guiado por un asistente que le permite ir resolviendo cada uno de los conflictos, mostrando en todo momento el directorio resultante, que se actualiza con cada nuevo conflicto resuelto. En el caso de conflictos de nombre permitirá elegir un nuevo nombre

para uno o los dos elementos involucrados en el conflicto, como puede verse en la Figura 2.

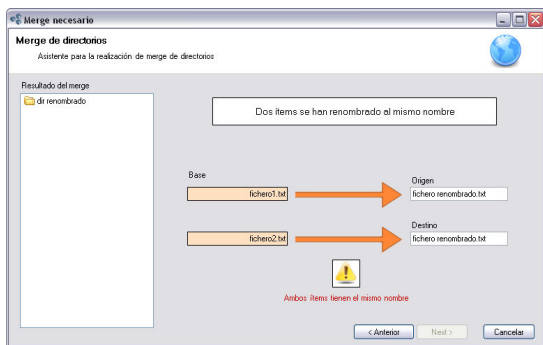


Figura 2. Conflicto de nombres, en el que dos elementos han sido renombrados a un mismo nombre, en un merge de directorios.

En el caso de que un mismo elemento haya sido renombrado con dos nombres distintos, permitirá elegir uno de los nombres con los que había sido renombrado o bien darle un nombre nuevo, como puede verse en la Figura 3:

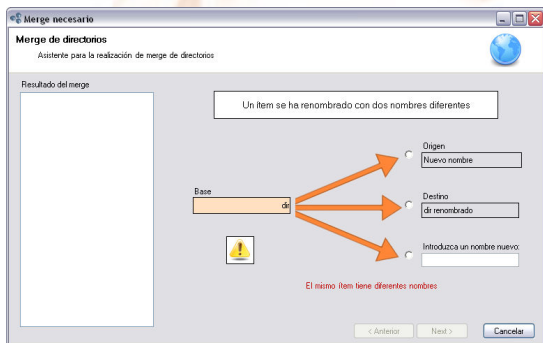


Figura 3. Conflicto de ítem, en el que un mismo ítem ha sido renombrado de dos formas distintas, en un merge de directorios.

Algunos de los nombres involucrados en los conflictos pueden dejar de ser válidos si otro conflicto ya les ha incorporado al directorio resultado (al no poder existir dos elementos con el mismo nombre). En ese caso, esos nombres tendrán que ser modificados en la resolución del conflicto.

Otros merges

En el caso del merge de los ficheros binarios, éste suele ser mucho más limitado. En la mayoría de los casos, el merge se limita a aquéllos que son completamente automáticos, en los cuales, sólo uno de los contribuidores ha modificado el fichero, permitiendo para los casos que no son automáticos la elección de uno de los contribuidores como resultado del merge. En el caso de Plastic SCM, cuando es necesario elegir uno de los contribuidores en un merge no automático, se muestra el contenido de los ficheros (ver Figura 4), siempre y cuando pueda ser representado. Esto se debe a que combinar dos ficheros binarios, combinando los cambios que se hayan realizado a nivel de byte, da casi siempre como resultado un fichero que ya no puede abrirse (que está roto). Para poder combinar los cambios entre dos ficheros binarios es necesario interpretar el formato de cada tipo de fichero y calcular los cambios y conflictos en función de cada uno de esos formatos, creando cambios que sean las diferentes acciones que se pueden hacer sobre ese tipo de fichero (cambiar un color para una imagen, añadir un párrafo para un documento, etc.). A esto, hay que añadir una dificultad adicional, y es que estos formatos suelen ser propietarios, muchos de los cuales además están cerrados, por lo que ni siquiera pueden ser estudiados. Es por esto por lo que las herramientas de merge binarias son tan limitadas.

Sin embargo, algunas herramientas son capaces de realizar el merge de sus propios ficheros. En estos casos si el control de versiones lo soporta pueden invocar a estas herramientas para que lo resuelvan. No obstante, en la mayor parte de los casos, estos merges suelen tener una potencia muy escasa, sin dar un resultado tan bueno como se desearía, sobre todo comparándolas con las herramientas de merge para ficheros de texto plano.

Un ejemplo de una herramienta muy usada y que es capaz de realizar el merge de sus propios ficheros es el Word. A diferencia de los casos de merge anteriores, el merge de Word es de dos vías: sólo se tienen en cuenta a los dos contribuidores para realizar el merge. En este caso, los cambios se calculan entre los dos contribuidores, en lugar de calcularse los cambios de cada uno de ellos sobre un antecesor común. Averiguar qué cambios

realizó cada fichero, y la tarea de decidir cuáles deben incorporarse y cuáles no resulta mucho más difícil para el usuario.

¿Quién realiza la integración?

La persona encargada de realizar la integración depende de la estrategia seguida. En una estrategia descentralizada será cada desarrollador el encargado de integrar sus cambios en el sistema. En cambio, en una estrategia centralizada será el integrador (o integradores) el responsable de integrar los cambios en el sistema.

Integración descentralizada

En esta estrategia de integración es el propio desarrollador el que se encargará de integrar sus cambios, el proceso de integración estará menos controlado y el desarrollador casi siempre tendrá una visión limitada del proyecto.

Al tener que ser cada uno de los desarrolladores, los encargados de realizar este proceso, es necesario que todos los desarrolladores posean un conocimiento más profundo del SCM. Deben conocer tanto el funcionamiento de la herramienta de SCM, como la estructura de SCM empleada en el proyecto (patrones de ramas, etiquetas, etc.).

Es necesario que el desarrollador tenga un mayor conocimiento del sistema. En caso de producirse algún conflicto, deberá decidir qué hacer no sólo con los cambios que él ha hecho, sino también con los cambios que los demás han introducido, incluso cuando estos cambios choquen entre sí.

La decisión sobre qué cambio ha de incorporarse en el sistema y cuál debe esperar o quedarse fuera, dependerá del desarrollador que ha implementado ese cambio.

Cada desarrollador deberá encargarse de asegurar la estabilidad del sistema tras la incorporación de sus cambios, así como de corregir los errores que puedan surgir.

Esta integración está soportada por todos y cada uno de los patrones de ramas explicados en entregas anteriores.

En este tipo de estrategia las integraciones suelen realizarse de forma continuada (integración continua). Cuando se emplea una integración continua, la creación de líneas de base (baselines: establecer una etiqueta a una versión concreta del código sobre la que se continuará trabajando) suele venir definida por los procedimientos de la empresa o realizarse en caso de necesidad. La aplicación de estas etiquetas la realizará una persona concreta dentro del proyecto, y no todos y cada uno de los desarrolladores al integrar.

Integración centralizada

En este tipo de integraciones el o los responsables de realizarlas serán los integradores.

El integrador es un nuevo rol que aparece dentro del desarrollo del proyecto. Puede ser una persona más del desarrollo que adicionalmente cumple este rol o ser una persona dedicada de forma exclusiva a esta labor.

El integrador es una persona con un mayor conocimiento del sistema que se está desarrollando y de la herramienta de SCM, siendo además una persona con experiencia en desarrollo y en el uso de herramientas SCM.

Gracias al uso de este rol, se exime al resto de los desarrolladores de este trabajo, así como de la necesidad de tener un mayor conocimiento del proyecto y de la herramienta de SCM utilizada.

Adicionalmente, la decisión de la incorporación de un cambio, o de su aplazamiento o descarte, se toma desde el punto de vista global al sistema que se está desarrollando, conjuntamente con las necesidades de cada momento.

En el caso de que una integración genere conflictos que el integrador no sea capaz de resolver por sí mismo, los resolverá con los desarrolladores implicados en el conflicto.

En esta estrategia de integración, las integraciones se realizarán de forma periódica. Al finalizar todas y cada una de las integraciones se creará una nueva línea de base y una nueva release. En todas y cada una de estas integraciones, el integrador se

encargará de asegurar la estabilidad e integridad de todo el sistema.

Esta estrategia no puede aplicarse en patrones de rama por proyecto.

Proceso de integración

El proceso de integración es muy dependiente del patrón de ramas que se esté empleando, por ello se explicará este proceso para cada uno de los patrones expuestos en entregas anteriores de forma independiente.

Rama por proyecto

Cuando un desarrollador termina sus cambios y ha comprobado que el sistema sigue siendo estable (mediante la política de pruebas existente) puede incorporar sus cambios protegiéndolos.

Cuando un desarrollador realiza la operación de proteger (check-in) es cuando incorpora sus cambios a la línea de desarrollo.

Si para alguno de los ficheros, con los que trabajaba localmente, se han creado nuevas versiones, necesitará hacerse un merge para combinar sus cambios con los que se hayan introducido en las mismas. Este merge se realizará desde la misma rama en la que está trabajando el usuario (que es la única en este patrón).

Después de realizar el merge, la versión desprotegida (en check-out) contendrá los cambios del usuario, más los que se habían realizado desde que él desprotegió el elemento, y será necesario volver a comprobar que el sistema sigue siendo estable antes de poder protegerlo de nuevo. Puede verse un esquema del proceso en la Figura 4:

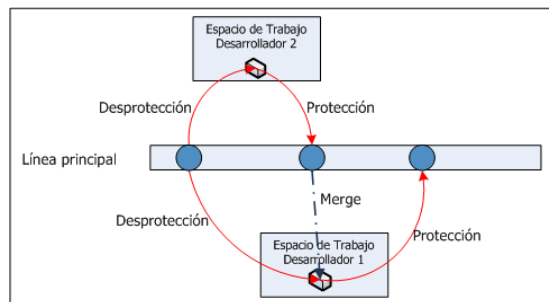


Figura 4. Esquema del proceso de integración en el patrón de rama por proyecto

Este patrón es muy apropiado para trabajar con un sistema de compilación continua, encargado de compilar el proyecto y pasar las pruebas de forma automatizada cada vez que un usuario protege un cambio. Todo este proceso se hace en un servidor de compilación dedicado. Para esto se utilizan aplicaciones como Continuum, CruiseControl o Anthill (para proyectos Java) o CruiseControl.Net (para proyectos .Net), que se encargan de controlar las ejecuciones, éstos a su vez emplean a otras herramientas como Ant o Maven (para proyectos Java), o Nant (para proyectos .Net) las cuales se encargan de realizar las compilaciones, ejecutar los tests y generar los informes.

Rama por release

En el caso de rama por release hay que diferenciar entre las integraciones dentro de una rama de release de las integraciones para propagar las correcciones de error entre ramas de release.

Para las integraciones dentro de cada una de las ramas de release, funcionará como el de rama por proyecto, pues dentro de cada release funciona como este patrón. Si por el contrario este patrón se combina con otro distinto al de rama por proyecto, las integraciones dentro de cada una de las ramas de release funcionarán como en ese otro patrón.

Para poder propagar los errores corregidos en una rama de release, hasta la rama de release actual, es necesario ir integrando esos errores hacia las ramas de release sucesivas.

En este patrón cada rama de release integrará los cambios de su release inmediatamente anterior. De esta manera asegura que todas las release posteriores a la de corrección del cambio lo tendrán incorporado.

Para realizar esta integración nos colocaremos en la última versión de la rama sobre la que deseamos integrar la corrección de error y haremos un merge desde la release, la cual ya tendrá corregido ese error.

Este merge, se encargará de buscar todos aquellos elementos que han sido modificados en la rama origen, desde la última vez que se integró. Esto es posible con herramientas de SCM que empleen los links de merge. En el caso de herramientas que no los tienen, es necesario que el usuario sepa y seleccione a mano los cambios que aún no hayan sido integrados (existen scripts para algunas de ellas, que tratan de automatizar esto a través de información adicional introducida en los comentarios).

Para cada uno de estos elementos se lanzará la herramienta de merge, la cual se encargará de combinar los cambios que los elementos han tenido en la rama origen, con los cambios que han tenido en la rama de destino.

Posteriormente, antes de proteger las versiones resultantes del merge se asegura la estabilidad del sistema mediante pruebas. Una vez que se verifique, se protegerían y quedarían integrados los cambios en la rama de release de destino.

Este proceso hay que repetirlo hasta que la rama de release de destino sea la actual, como puede verse en la Figura 5:

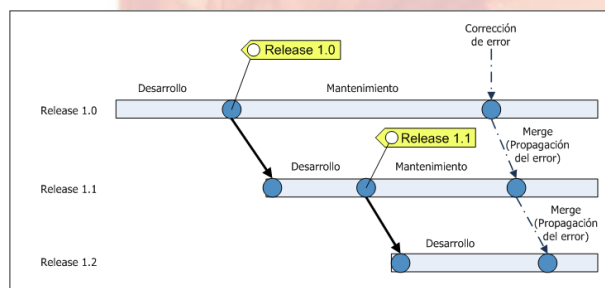


Figura 5. Esquema del proceso de integración en el patrón de rama de release

Estas integraciones pueden hacerse de forma centralizada o descentralizada.

En el caso de las descentralizadas, cada desarrollador al corregir un error tendría que encargarse de propagar esa corrección hasta la rama de release actual.

En el caso de las centralizadas, será el integrador el que periódicamente se encargue de propagar todas las correcciones que se hayan hecho a la release actual. Esto tiene una clara ventaja, y es que, al incorporar los cambios que se hayan corregido de la release más antigua a la actual, se van incorporando las correcciones introducidas en las releases intermedias. De esta forma, para integrar todos los cambios, en vez de tener que realizar tantas propagaciones como cambios sólo se hace una.

Rama de mantenimiento

En el caso de este patrón también es necesario distinguir las integraciones dentro de la rama de mantenimiento y la principal (la de desarrollo), de las integraciones entre la rama de mantenimiento y la principal para incorporar una corrección de error.

El trabajo dentro de las ramas de mantenimiento y de la principal viene determinado por el patrón con el que se esté combinando (en caso de no ser ninguno, funciona como el de rama por proyecto).

En el caso de incorporar una corrección de error desde la rama de mantenimiento a la principal, el proceso es muy similar a como funciona la incorporación de un cambio de una release a otra en el de rama por release. En este caso, será necesario colocarnos en la rama principal y hacer un merge desde la rama de mantenimiento en la que se ha corregido el error. La herramienta se encargará de buscar qué elementos han sido cambiados desde la última integración, que son aquellos que participan en la corrección del error. En el caso de que la herramienta no posea links de merge que le permita buscar estos elementos, tendrá que ser el usuario el que se encargue de seleccionarlos.

Posteriormente para aquellos elementos que necesiten ser mezclados se lanzará la herramienta de merge, al igual que en el caso anterior.

También en este caso antes de confirmar la incorporación de esos cambios en la rama principal, será necesario verificar el sistema mediante las pruebas. Si el sistema permanece estable, se confirman los cambios protegiéndolos, quedando integrados en la rama principal. Puede verse un esquema de este proceso en la Figura 6.

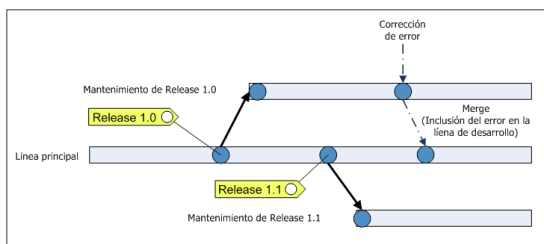


Figura 6. Esquema del proceso de integración en el patrón de rama de mantenimiento

Este patrón también permite trabajar con cualquiera de las estrategias de integración expuestas.

Rama por desarrollador

En este patrón, cuando un desarrollador termine de hacer un cambio en su rama, debería hacerse un rebase (se explicará más adelante) de la rama principal antes de integrarlo.

Tras hacerse un rebase, la rama del desarrollador contendrá la última versión del proyecto que esté confirmada, más sus cambios.

Con la versión de código en la rama ya actualizada con el contenido de la rama principal, se debe comenzar a probar (siguiendo la política de prueba establecida).

En caso de que de algún error, éste puede ser corregido en la rama sin que los demás se vean afectados.

Una vez se haya confirmado que el sistema sigue siendo estable con los nuevos cambios, hay que integrarlo en la rama principal. Para ello nos colocaremos en la rama principal y realizaremos un merge desde la rama del desarrollador. En caso de que la herramienta de SCM disponga de

links de merge, este merge será sumamente sencillo, pudiendo dar conflictos solamente con aquellos cambios que se hayan incorporado desde que se hizo el rebase hasta que se integren los cambios ya probados.

Una vez integrados los cambios en la rama principal, la rama principal debe ser sometida a pruebas (según estén definidas en la política de pruebas). Estas pruebas pueden o no ser las mismas a las que se sometió a la rama del desarrollador. Es importante asegurar la estabilidad de la rama principal en todo momento. Sólo una vez que se ha confirmado esto, los cambios deben protegerse pasando a formar parte de la rama principal y a ser visibles para el resto de los desarrolladores.

Aunque el rebase no es necesario antes de integrar la rama, éste nos facilita mucho la integración. Además mantiene una mayor estabilidad de la rama principal, porque al probar los cambios de la rama que vamos a integrar conjuntamente con los últimos cambios incorporados en la rama principal, aseguramos que los cambios de la rama funcionan correctamente con la última versión del producto, y que al integrarse en la rama principal no fallen.

La última parte del proceso, la de integrar los cambios en la rama principal una vez probados y las posteriores pruebas de la rama principal sería el proceso de integración propiamente dicho. Este puede realizarse por un integrador, en caso de usar una estrategia centralizada, o por el propio desarrollador al que pertenece la rama, en el caso de utilizar una estrategia descentralizada. Puede verse un esquema de este proceso en la Figura 7.

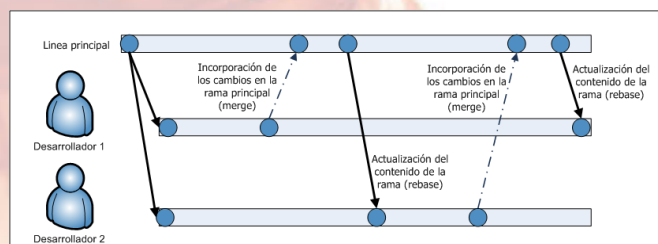


Figura 7. Esquema del proceso de integración en el patrón de rama por desarrollador

Rama por tarea

Antes de dar una tarea por cerrada, ésta debe haber pasado las pruebas necesarias para darla por válida (según la política de pruebas). Una tarea cerrada puede ser integrada en la rama principal, y en algunos casos en otras ramas (ramas de mantenimiento generalmente).

Para integrar la tarea, nos colocaremos en la rama principal o en aquella rama en la que deseamos integrarla y realizaremos el merge, tal y como hacíamos en los casos anteriores. Una vez resueltos los conflictos y pasadas las pruebas, se confirmarán los cambios quedando incluidos en la rama principal, tal y como puede verse en la Figura 8.

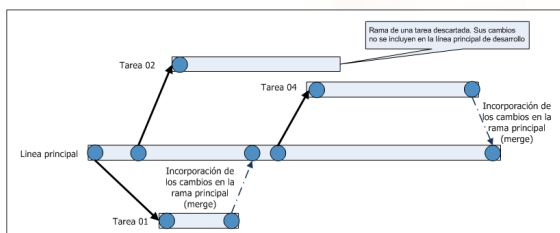


Figura 8. Esquema del proceso de integración en el patrón de rama por tarea

Es importante tener en cuenta que con esta metodología el número de ramas creadas, así como el de merge realizados es muy elevado. Por ello es importante que la herramienta de control de tareas utilizada tenga un potente soporte de ramas y que incluya links de merge.

La integración en este patrón puede ser realizada por el desarrollador que concluyó la tarea, si empleamos una estrategia de integración descentralizada, o por el integrador, en caso de usar una estrategia de integración centralizada. No obstante, en este patrón suele ser habitual el uso de una estrategia centralizada.

Rebase

Cuando se trabaja en una rama diferente a la rama principal, es posible que mientras se esté trabajando de forma aislada, en la

rama principal se introduzcan nuevos cambios. Para incorporar esos nuevos cambios a nuestra rama de trabajo, se emplea el rebase, tal y como puede verse en la Figura 9.

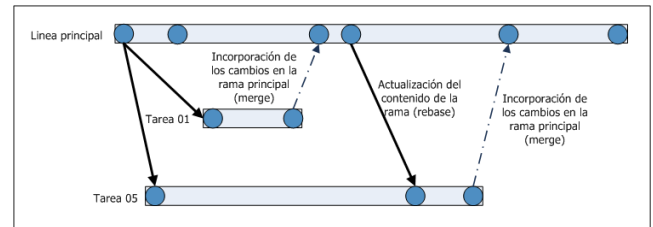


Figura 9. Utilización del rebase en un patrón de rama por tarea

El rebase no es más que una integración desde la rama principal de desarrollo a la rama actual del usuario, para actualizar el contenido de la rama con la última versión de código.

Este proceso se realiza siempre que en la rama principal se genera una nueva release, o cuando necesitemos incorporar en nuestra rama de trabajo un cambio que haya sido introducido en la línea principal.


Este proceso es realizado por el desarrollador de la rama. Para ello estando en la rama actual de trabajo se realiza un merge desde la rama principal.

Al igual que sucedía en los casos anteriores, si la herramienta de SCM empleada no posee links de merge, este proceso se complica. Esto es así porque si se ha realizado algún rebase con anterioridad sobre la misma rama, será el propio usuario el que tenga que seleccionar los elementos que han sido modificados desde el anterior rebase en la rama principal.

El uso de la operación de rebase es muy frecuente en el patrón de rama por desarrollador. También se emplea en el de rama por tarea, cuando una tarea requiere más tiempo que el tiempo que se tarda en generar una nueva release en la rama principal.


Conclusiones

A lo largo de esta serie de artículos se ha introducido al usuario en las herramientas de SCM. Se han introducido los términos usados



en el uso de un SCM, así como sus principales elementos. Se han explicado las ventajas de emplear un SCM y los errores y problemas que se evitan con su uso. Se han mostrado diferentes escenarios en los que se emplean este tipo de herramientas. Se han aportado pautas y guías para el uso de estas herramientas y, se han aportado algunos consejos para su correcto uso y para poder sacar el máximo provecho de ellas, mostrando algunas de las problemáticas del uso inadecuado de un control de versiones.

En este último artículo se ha mostrado al usuario el funcionamiento de una de las operaciones más críticas dentro de un control de versiones: el merge. Asimismo, se ha tratado de guiar al usuario en los procesos de integración y rebase, procesos muy habituales en el uso de un control de versiones y unos de los más complejos y problemáticos. Se ha tratado de dar pautas para facilitar su uso y asegurar la calidad y estabilidad del sistema durante estos procesos.



Borja Ruiz Arroyo
Francisco José García Peñalvo
Artículo publicado en la revista
Solo Programadores
Marzo 2007