**Control de Versiones**

Un *sistema de control de versiones* (o *sistema de control de revisiones*) es una combinación de tecnologías y practicas para seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web. Si nunca antes se ha utilizado un control de versiones, lo primero que hay que hacer es conseguir a alguien que sí lo haya hecho y hacer que se una al proyecto. Hoy en día todo el mundo espera que al menos el código fuente del proyecto este bajo un control de versiones y probablemente no se tomen el proyecto seriamente si no se utiliza este sistema con un mínimo de competencia.

La razón por la cual el control de versiones es universal es porque ayuda virtualmente en todos los aspectos al dirigir un proyecto: comunicación entre los desarrolladores, manejo de los lanzamientos, administración de fallos, estabilidad entre el código y los esfuerzos de desarrollo experimental y atribución y autorización en los cambios de los desarrolladores. El sistema de control de versiones permite a una fuerza coordinadora central abarcar todas estas áreas. El núcleo del sistema es la *gestión de cambios*: identificar cada cambio a los ficheros del proyecto, anotar cada cambio con meta-data como la fecha y el autor de la modificación y disponer esta información para quien sea y como sea. Es un mecanismo de comunicación donde el cambio es la unidad básica de información.

Aun no hemos discutido todos los aspectos de utilizar un sistema de control de versiones ya que es un tema tan extenso que será introducido según el tópico a lo largo de este libro. Ahora, vamos a concentrarnos en seleccionar y configurar un sistema de control de versiones de forma que fomentemos un desarrollo cooperativo.

**Vocabulario**

En este libro no podemos enseñar como utilizar el control de versiones si nunca antes lo ha utilizado, pero sería imposible continuar sin conocer algunos términos clave. Estos son útiles independientemente del sistema particular utilizado: son definiciones básicas y verbos sobre la colaboración en red y serán utilizados a lo largo del libro. Incluso si no existiera ningún sistema de control de versiones, el problema del control de los cambios aun existiría y estas palabras nos dan un lenguaje para hablar acerca de este problema consistentemente.

**"Versión" Versus "Revisión"**

El termino *versión* es a veces utilizado como un sinónimo para "revisión", pero aquí no voy a utilizarla de esta forma, ya que se puede confundir fácilmente con "versión" en el sentido de una versión de un programa—así que, el número de lanzamiento o edición como en "Versión 1.0". Y aunque la frase "control de versiones" es un estándar, continuare utilizándolo como sinónimo para "control de revisiones" y para "control de cambios".

*commit*

**Realizar un cambio en el proyecto**. Formalmente, almacenar un cambio en la base de datos del control de versiones de forma que pueda ser incorporado en lanzamientos futuros del proyecto. "Commit" puede ser utilizado como un verbo o como un sustantivo. Como un sustantivo, es esencialmente un sinónimo de "cambio". Por ejemplo: "He commited una reparación para un fallo reportado en Mac OS X que hacía que el servidor se colgara. Jóse ¿podrías por favor revisarlo y verificar que no estoy haciendo mal la asignación?"

*Mensaje de registro*

Un **pequeño comentario añadido a cada commit que describe el tipo y el propósito del commit**. Los mensajes de registro forman parte de los documentos más importantes de cualquier proyecto ya que son un puente entre el lenguaje altamente técnico de los cambios individuales en el código y el lenguaje más orientado al usuario de características, resolución de fallos y progreso del proyecto. Más adelante vamos a ver la forma de distribuir estos mensajes a la audiencia apropiada y también [“Tradición en la organización del contenido”](http://producingoss.com/es/growth.html#codifying-tradition) en [Capítulo 6, *Comunicaciones*](http://producingoss.com/es/communications.html) discutimos como ENCOURAGE a los voluntarios para que escriban mensajes de registro útiles y concisos.

*update*

Solicitar los cambios (commits) que han realizado otros en la copia local del proyecto, esto **actualiza esta copia a la ultima versión**. Es una operación muy común ya que la mayoría de los desarrolladores actualizan su código varias veces al día y así saben que están ejecutando casi lo mismo que los otros desarrolladores, así que si se descubre un fallo es muy posible que este aun no haya sido resuelto. Por ejemplo: "Hey, he notado que el código del índice está fallando en el último byte. ¿Es esto un nuevo fallo?" "Sí, pero fue resuelto la semana pasada—prueba actualizar para resolverlo."

*repositorio*

Una **base de datos en la que los cambios son almacenados**. Algunas versiones de sistemas de control de versiones son **centralizados, es decir, existe un único repositorio maestro, el cual almacena todos los cambios en el proyecto**. **Otros sistemas son descentralizados, cada desarrollador tiene su propio repositorio y los cambios pueden ser intercambiados entre repositorios arbitrariamente**. El sistema de control de versiones mantiene un registro de las dependencias entre estos cambios y cuando llega el momento de realizar un lanzamiento, un conjunto particular de cambios es aprobado para ese lanzamiento. La cuestión de cual sistema es mejor es otra de las guerras santas del desarrollo de software. Intentad no caer en la trampa de discutir sobre esto en las listas de correo del proyecto.

*checkout*

**El proceso de obtener una copia del proyecto desde el repositorio**. Por lo general, un checkout produce un árbol de directorios llamado "copia funcional" desde el cual los cambios serán enviados de vuelta al repositorio original. En algunas versiones descentralizadas de sistemas de control, cada copia funcional es en si mismo un repositorio y los cambios son empujados (o atraídos) a cualquier repositorio que este dispuesto a aceptarlos.

*copia funcional*

El árbol de directorios privado de cada desarrollador que contiene el código fuente del proyecto y posiblemente las páginas web u otros documentos. Una copia funcional también contiene una pequeña cantidad de meta-data administrada por el sistema de control de versiones, informando a la copia funcional cual es repositorio central de procedencia, la revisión de los ficheros presentes, etc**. Generalmente, cada desarrollador tiene su propia copia funciona en la cual realiza y prueba los cambios y desde la cual envía sus commits (cambios)**.

*revisión*, *cambio*, *conjunto de cambios*

Una revisión es usualmente una encarnación específica de un fichero o directorio en particular. Por ejemplo, si el proyecto se inicia en la revisión 6 del fichero F y alguien envía un cambio al fichero F, esto produce la revisión 7 de F. Algunos sistemas también utilizan revisión (revision), cambio (change) o conjunto de cambios (changeset) para referirse a un conjunto de cambios enviados juntos como una unidad conceptual.

Estos conceptos pueden tener distintos significados técnicos en cada sistema de control de versiones, pero en general, la idea es siempre la misma: dar un sistema para comunicar de manera precisa la historia de cambios en un fichero o conjunto de ficheros (inmediatamente antes y después de que se ha corregido un fallo). Por ejemplo: "Eso se ha resuelto en la revisión 10" o "Se ha corregido eso en la revisión 10 del fichero foo.c."

Cuando se habla sobre ficheros o una colección de ficheros sin especificar una revisión en particular, por lo general se asume que nos referimos a la revisión disponible más reciente.

*diff*

Una representación contextual de un cambio. Un diff **muestra las lineas que han sido modificadas**, como y además, algunas lineas contextuales rodeándolas a cada lado. Un desarrollador familiarizado con el código puede, con leer un diff de ese código, entender lo que hace el cambio e incluso detectar fallos.

*etiqueta (tag)*

Una etiqueta para una colección particular de ficheros en una revisión específica. Los tags son utilizados para preservar capturas interesantes del proyecto. Por ejemplo, un tag es hecho para cada lanzamiento público, de forma que cada persona pueda obtener, directamente desde el sistema de control de versiones, el conjunto exacto de ficheros/revisiones que componen el lanzamiento. Algunos tags comunes son como Release\_1\_0, Delivery\_00456, etc.

*rama (branch)*

Es una **copia del proyecto**, bajo el control de versiones, **pero aislado, de forma que los cambios realizados en esta rama no afecten al resto del proyecto** y vice versa, **excepto cuando los cambios sean deliberadamente "unidos" de un lado al otro**. **Las ramas también son conocidas como "lineas de desarrollo".** Incluso cuando un proyecto no tiene ramas específicas se considera que el desarrollo se esta produciendo en la rama principal, también conocida como "línea primaria" o "*trunk*".

Las ramas o branches, permiten aislar diferentes lineas de desarrollo de si mismas. Por ejemplo, una rama puede ser utilizada para un desarrollo experimental que sería demasiado inestable para la rama principal. O al contrario, una rama puede ser utilizada como sitio para estabilizar una versión para lanzamiento. Durante el proceso de lanzamiento, el desarrollo regular se mantendría ininterrumpida en la rama principal. Mientras tanto, en la rama de lanzamiento, ningún cambio es aceptado excepto aquellos aprobados por el responsable del lanzamiento. De esta manera, realizar un lanzamiento no tiene porque interferir con el trabajo de desarrollo que se está realizando. Para más información [“Las ramas para evitar cuellos de botella”](http://producingoss.com/es/vc.html#branches) más adelante en el capítulo para una discusión más detallada sobre las ramas.

*merge*

**Mover un cambio de una rama a otra, lo que incluye unir desde la rama principal a otra rama o vice versa**. De hecho, estos son las uniones más comunes y es rara la ocasión en la que esto se hace entre dos ramas no principales. Para más información sobre los merge [“Singularidad de la información”](http://producingoss.com/es/vc.html#vc-singularity).

**"Merge" tiene otro significado: es lo que hace el sistema de control de versiones cuando se encuentra con que dos personas han realizado cambios en un mismo fichero sin relación alguna**. **Ya que estos cambios no interfieren entre ellos, cuando alguna de estas personas actualizan su copia del fichero (el cual ya contiene los cambios) los cambios de la otra persona serán unidos automáticamente**. Esto sucede muy a menudo, especialmente en proyectos con múltiples personas realizando cambios en el mismo código. **Cuando dos cambios diferentes están relacionados, el resultado es un "conflicto".**

*conflicto*

**Sucede cuando dos o más personas intentan realizar diferentes cambios en la misma porción de código**. Todos los sistemas de control de versiones detectan estos conflictos automáticamente y notifican a al menos uno de los humanos involucrados de que sus cambios entran en conflicto con los de alguien más. Es entonces tarea de esta personas *resolver* el conflicto y comunicar esa resolución al sistema de control de versiones.

*bloqueo (lock)*

**Declaración de un intento exclusivo para cambiar un fichero o directorio en particular**. Por ejemplo, "No puedo enviar cambios a las paginas web ahora mismo, ya que parece que Alfredo las tiene bloqueadas mientras arregla sus imágenes de fondo." No todos los sistemas de control de versiones ofrecen la posibilidad del bloqueo y aquellos que sí lo permiten, no es necesario que se utilice. Esto es porque el desarrollo paralelo y simultaneo es la norma y bloquear a la gente para que no puedan modificar los ficheros es contrario a la idea del sistema.

**El modelo del sistema de control de versiones que requiere el bloqueo de ficheros suele ser llamado *bloqueo-modificación-desbloqueo* y el modelo que no requiere del bloqueo es llamado *copia-modificación-unión*.** Una excelente explicación en profundidad y comparaciones puede ser encontrada en <http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html>. En general, el modelo de copia-modificación-unión es el mejor para el desarrollo open source y todos los sistemas de control de versiones discutidos en este libro soportan este modelo.