ÍNDICE

1. Introducción

Cuando estamos desarrollando software, el código fuente está cambiando continuamente, bien por el propio desarrollo o por el mantenimiento a que se ve sometido. Además, los proyectos en ocasiones se desarrollan por fases o se hacen diferentes entregadas al cliente. Todos estos factores hacen necesario un sistema de control de versiones.

Las ventajas de utilizar un sistema de control de versiones son múltiples. Un sistema de control de versiones bien diseñado facilita al equipo de desarrollo su labor, permitiendo que varios programadores trabajen en el mismo proyecto (incluso sobre los mismos archivos) de forma simultánea, permitiendo gestionar los conflictos que se puedan producir por actualizaciones simultaneas sobre el mismo código. Las herramientas de control de versiones proveen de un sitio central donde almacenar el código fuente de la aplicación, así como el historial de cambios realizados a lo largo de la vida del proyecto.

También permite a los desarrolladores volver a versiones estables previas del código fuente si es necesario. Una versión, desde el punto de vista de la evolución, se define como la forma particular de un objeto en un instante o contexto dado. Se denomina revisión, cuando se refiere a la evolución en el tiempo. Pueden coexistir varias versiones alternativas en un instante dado y hay que disponer de un método, para designar las diferentes versiones de manera organizada.

Existen distintas alternativas de control de versiones de código abierto, **GIT**, **CVS**, **Subversion**, **Mercurial**... En ocasiones podrán ser motivo de gestión proyectos software en desarrollo o simplemente conjuntos de archivos de algún otro uso.

Un repositorio interesante para la gestión de proyectos es **Bitbucket**, está basado en Git y dispone de una versión gratuita para equipos de 5 personas. Ideal para gestionar el proyecto fin de ciclo.

2. Tipos de herramientas de control de versiones.

De acuerdo al modo de organizar la información, se pueden clasificar las herramientas de control de versiones como:

- **Sistemas locales**: se trata del control de versiones donde la información se guarda en diferentes directorios en función de sus versiones. Toda la gestión recae sobre el responsable del proyecto y no se dispone de herramientas que automaticen el proceso. Es viable para pequeños proyectos donde el trabajo es desarrollado por un único programador.

- **Sistemas centralizados**: responden a una arquitectura cliente-servidor. Un único equipo tiene todos los archivos en sus diferentes versiones, y los clientes replican esta información en sus entornos de trabajo locales. El principal inconveniente es que el servidor es un dispositivo crítico para el sistema ante posibles fallos.

- **Sistemas distribuidos**: en este modelo cada sistema hace con una copia completa de los cheros de trabajo y de todas sus versiones. El rol de todos los equipos es de igual a igual y los cambios se pueden sincronizar entre cada par de copias disponibles. Aunque técnicamente todos los repositorios tienen la posibilidad de actuar como punto de referencia; típicamente a nivel funcionan, uno será el repositorio principal y el resto asumirán un papel de clientes sincronizando sus cambios con éste.

3. Estructura de las herramientas de control de versiones.

Una herramienta de control de versiones es un sistema de mantenimiento de código fuente (o de grupos de archivos en general) de gran utilidad para grupos de desarrolladores que trabajan cooperativamente.

Los clientes pueden actuar simultáneamente sobre los mismos ficheros, comparar diferentes versiones de ficheros, solicitar una historia completa de los cambios, o sacar una "foto" histórica del proyecto tal como se encontraba en una fecha o versión determinada.

Los sistemas de control de versiones suelen estar formados por un conjunto de componentes:

- **Repositorio**: es el lugar de almacenamiento o base de datos de los proyectos.
- **Rama**: revisiones paralelas del contenido del proyecto para efectuar cambios sin tocar la evolución principal (que normalmente se refiere como rama principal o trunk).
- **Versión**: fotografía del estado de una de las revisiones de alguna rama del repositorio en un determinado momento en el tiempo.
- **Revisión**: evolución en el tiempo de las ramas del proyecto.
- **Etiqueta**: información textual que se añade a un conjunto de archivos o a un módulo completo para indicar alguna información importante. En ocasiones identifican versiones del producto ya cerradas.

Algunos de los servicios que típicamente proporcionan son:

- **Creación de repositorios**. Creación del esqueleto de un repositorio sin información inicial del proyecto.
- **Clonación de repositorios**. La clonación crea un nuevo repositorio y vuelca la información de algún otro repositorio ya existente. Crea una réplica.
- **Descarga de la información del repositorio principal al local**. Sincroniza la copia local con la información disponible en el repositorio principal.
- **Carga de información al repositorio principal desde el local**. Actualiza los cambios realizados en la copia local en el repositorio principal.
- **Gestión de conflictos**. En ocasiones, los cambios que se desean consolidar en el repositorio principal entran en conflicto con otros cambios que hayan sido subidos por algún otro desarrollador. Cuando se da esta situación, las herramientas de control de versiones tratan de combinar automáticamente todos los cambios. Si no es posible sin pérdida de información, muestra al programador los conflictos acontecidos para que sea éste el que tome la decisión de como combinarlos.
- **Gestión de ramas**. Creación, eliminación, integración de diferencias entre ramas, selección de la rama de trabajo.
- **Información sobre registro de actualizaciones**.
- **Comparativa de versiones**. Genera información sobre las diferencias entre versiones del proyecto.

4. Herramientas de control de versiones.

Algunas de las herramientas con amplia presencia son:

- **GIT**: herramienta de control de versiones diseñada por Linus Torvalds. Trabajaremos con ella a lo largo del tema.
- **Subversion - SVN**: herramienta de control de versiones de código abierto distribuida bajo licencia de tipo Apache/BSD.
- **Visual Studio Team Foundation Server**: producto de Microsoft que ofrece control de código fuente, recolección de datos, informes y seguimiento de proyectos.
- **Darcs**: sistema de control de versiones distribuido creado por David Roundy.
- **Mercurial**: herramienta multiplataforma de software libre desarrollada por Matt Mackall. Esta aplicación está principalmente diseñada para ser utilizada desde la línea de comandos.

5. Planificación de la gestión de configuraciones

La **Gestión de Configuraciones del Software** (GCS) es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del ciclo de vida de un producto.

Una configuración puede cambiar porque se añaden, eliminan o se modifican elementos. También puede cambiar, debido a la reorganización de los componentes, sin que estos cambien.

La planificación de la **Gestión de Configuraciones del Software**, está regulado en el estándar **IEEE 828**.

La Gestión de Configuraciones de Software se va a componer de cuatro tareas básicas:

1. **Identificación**. Se trata de establecer estándares de documentación y un esquema de identificación de documentos.
2. **Control de cambios**. Consiste en la evaluación y registro de todos los cambios que se hagan de la configuración software.
3. **Auditorías de configuraciones**. Junto con las revisiones técnicas formales, se utilizan para garantizar que el cambio se ha implementado correctamente.
4. **Generación de informes**. El sistema software está compuesto por un conjunto de elementos, que evolucionan de manera individual, por consiguiente, se debe garantizar la consistencia del conjunto del sistema.

6. Gestión del cambio.

Las herramientas de control de versiones no garantizan un desarrollo razonable, si cualquier componente del equipo de desarrollo de una aplicación puede realizar cambios e integrarlos en el repositorio sin ningún tipo de control.

Para garantizar que siempre disponemos de una línea base para continuar el desarrollo, es necesario aplicar controles al desarrollo e integración de los cambios. **El control de cambios es un mecanismo que sirve para la evaluación y aprobación de los cambios hechos a los elementos de configuración del software.**

Pueden establecerse distintos tipos de control:

- **Control individual, antes de aprobarse un nuevo elemento**. Cuando un elemento de la configuración está bajo control individual, el programador responsable cambia la documentación cuando se requiere. El cambio se puede registrar de manera informal, pero no genera ningún documento formal.

- **Control de gestión u organizado, conduce a la aprobación de un nuevo elemento**. Implica un procedimiento de revisión y aprobación para cada cambio propuesto en la configuración. Como en el control individual, el control a nivel de proyecto ocurre durante el proceso de desarrollo, pero es usado después de que haya sido aprobado un elemento de la configuración software. El cambio es registrado formalmente y es visible para la gestión.

- **Control formal, se realiza durante el mantenimiento**. Ocurre durante la fase de mantenimiento del ciclo de vida software. El impacto de cada tarea de mantenimiento se evalúa por un Comité de Control de Cambios, el cuál aprueba las modificaciones de la configuración software.