Sistemas de Control de Versiones De Código Fuente

Se llama *control de versiones de código fuente* a sistemas que permiten la gestión de los diversos cambios que se realizan sobre los elementos del código fuente de algún software o una configuración del mismo. Una versión, revisión o edición de un producto de software, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque se puede realizar de forma manual (ej: copiando todo el código fuente con un nombre de versión diferente cada vez) es aconsejable utilizar herramientas como Git, Subversion, Mercurial, etc.

Características

Un sistema de control de versiones debe proporcionar:

- Un mecanismo de almacenamiento del código fuente (archivos de texto, imágenes, documentación...) a gestionar.
- La posibilidad de realizar cambios sobre los archivos almacenados (modificaciones parciales, añadir, borrar, renombrar o mover archivos).
- Un registro histórico de las acciones realizadas sobre cada archivo (y quién las hizo), normalmente pudiendo volver a un estado anterior del software.

Repositorio

Es el lugar donde se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor.

Incluso en sistemas distribuídos como Git, se suele utilizar un repositorio "central" o "maestro" al que contribuyen todos los desarrolladores de un equipo.

Módulo

Es el conjunto de directorios y/o archivos dentro del repositorio que pertenecen a un proyecto común.

Un proyecto de software se puede dividir en módulos (*modularizar*) para facilitar su desarrollo (Ej. dividir las distintas tareas entre los distintos equipos) y cada módulo estar contenido dentro de un mismo repositorio para todo el proyecto.

Revisión (*Version*)

Una revisión es una versión determinada de la información que se gestiona.

Algunos sistemas identifican las revisiones con un contador (Ej. Subversion). Otros sistemas las identifican mediante un código de detección de modificaciones (Ej. Git usa un *hash* SHA1).

A la última versión se le suele identificar de forma especial con el nombre de **HEAD** (cabeza).

Se puede marcar una revisión concreta usando rótulos o tags.

Rotular (*Tag*)

Es "marcar" alguna versión de cada uno de los archivos del módulo desarrollado en un momento preciso, con un nombre común ("etiqueta" o "rótulo", *tag*) para poder "volver" a dicho estado usando ese nombre.

Hay varios usos de los tag, que varían de acuerdo a la metodología de desarrollo del equipo, pero en general es práctica común marcar las revisiones importantes del proyecto con un tag, por ejemplo las revisiones publicadas (*1.0*, *3.8.4*, etc.)

En algunos sistemas se considera un tag como una rama en la que los archivos no evolucionan, están "congelados".

Rama (*Branch*) o Bifurcación

En cualquier momento en el tiempo, un módulo puede ser bifurcado, por ejemplo para investigar una posibilidad de implementación de una característica deseada sin "perturbar" el desarrollo principal del módulo.

En dicho instante se tienen dos copias (ramas) que pueden evolucionar de manera independiente y eventualmente se pueden "fusionar" (*merge*) una vez que se decide implementar esa función con ese código.

Es común la creación de ramas para el desarrollo de funcionalidades nuevas (*feature branch*), para la publicación de una nueva versión del software (*release branch*), o para la corrección de errores (*hotfix branch*).

Check-Out

El uso de este término varía un poco de acuerdo al sistema de VCS.

En los sistemas centralizados (Ej. Subversion) es realizar una copia del repositorio en la máquina local del desarrollador, y se realiza una única vez. En los sistemas distribuídos (Ej. Git) es crear una rama, o cambiar a una rama, y se realiza varias veces durante el desarrollo (a menudo, varias veces en el día).

Se puede hacer check-out de una rama específica, de un tag específico, o (normalmente) de *HEAD*, es decir, de la versión actual.

Check-In (*commit*, *push*, *submit*...)

Es la acción de enviar al repositorio los cambios que el desarrollador hizo en su copia local del proyecto, para su integración (*luego de haberse asegurado por medio de pruebas que todo funciona correctamente*...).

De esta manera, dichos cambios estarán disponibles para el resto del equipo de desarrollo.

Conflicto

Los conflictos ocurren cuando el sistema de control de versiones no puede manejar adecuadamente los cambios realizados por dos o más usuarios en un mismo archivo.

El desarrollador que obtiene el conflicto (Ej. Al intentar hacer *check-in* de su copia local) es quien debe resolverlo combinando los cambios, o eligiendo uno de ellos para descartar el otro.

Fusión (*Merge*)

Una integración o fusión une dos conjuntos de cambios sobre un archivo, o un conjunto de archivos, en una revisión unificada de dicho archivo o conjunto de archivos.

Los sistemas puden hacer las fusiones de manera automática, pero cuando no se puede se genera un conflicto que algún desarrollador deberá resolver de forma manual, y luego fusionar la versión resuelta.

Si las ramas no evolucionan durante mucho tiempo de manera independiente de la línea de desarrollo principal (*trunk*), en general el sistema las fusiona automáticamente.

Arquitecturas de Almacenamiento

- **Distribuído**: cada desarrollador tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es común (aunque no obligatorio) el uso de un repositorio "centralizado" como punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.
- **Centralizado**: existe un repositorio centralizado de todo el código. Las tareas administrativas son más fáciles a costa de la flexibilidad, porque todas las decisiones importantes (Ej. crear una rama nueva) las debe aprobar el responsable. Ejemplos: CVS, Subversion y Team Foundation Server.

Ventajas Sistemas Distribuídos

- Menor concectividad a la red para operar, aumenta la autonomía del desarrollador y la velocidad de desarrollo.
- Los desarrolladores pueden trabajar aunque no esté disponible el repositorio remoto (si lo hubiera).
- Como los repositorios locales son una réplica del centralizado, la información está disponible en varios lugares lo que facilita la recuperación del sistema en caso de desastre en el repositorio central. Disminuye la necesidad de backups (*aunque no la elimina*).

Ventajas Sistemas Distribuídos - Cont.

- Permite mantener repositorios centrales "más limpios" en el sentido de que un desarrollador puede decidir que ciertos cambios locales no son relevantes para el resto y no enviarlos.
- El servidor remoto requiere menos recursos que los que necesitaría un servidor centralizado ya que gran parte del trabajo lo realizan los repositorios locales.
- Se facilita el uso de las ramas (creación, evolución y fusión) y se aprovecha al máximo su potencial. Se pueden crear ramas en el repositorio remoto para corregir errores o crear funcionalidades nuevas. Pero también se pueden crear ramas en los repositorios locales para que los desarrolladores prueben cosas y luego de realizar pruebas integren sus cambios al repositorio remoto.

Ventajas Sistemas Centralizados

- En los sistemas distribuidos hay menos control a la hora de trabajar en equipo ya que no se tiene una versión centralizada de todo lo que se está haciendo en el proyecto.
- En los sistemas centralizados las versiones están identificadas por un número de versión. Sin embargo en los sistemas de control de versiones distribuidos no hay números de versión, ya que cada repositorio tendría sus propios números de revisión dependiendo de los cambios. En lugar de eso, cada versión tiene un identificador al que se le puede asociar una etiqueta (*tag*).

Flujo de Trabajo Centralizado

- Cada desarrollador es un nodo de trabajo. Por otro lado hay un repositorio remoto central que funciona a modo de punto de sincronización. Todos los nodos de trabajo operan en pie de igualdad sobre el repositorio remoto central.
- Una desventaja es que si dos desarrolladores clonan desde un punto central, y ambos hacen cambios; sólo el primero que envíe sus cambios lo podrá hacer limpiamente. El segundo deberá fusionar previamente su trabajo con el del primero, antes de enviarlo, para evitar sobreescribir los cambios del primero. Es necesario hacer un paso previo ya que no se pueden enviar cambios no directos (*non-fast-forward changes*).

Flujo de Trabajo con Gestor de Integraciones

- Cada desarrollador tiene acceso de escritura a un repositorio propio público y acceso de lectura a los repositorios públicos de todos los demás desarrolladores. Por otro lado hay un repositorio canónico, representante "oficial" del proyecto.
- Para contribuir en estos proyectos cada desarrollador crea su propio clon público del repositorio canónico y envía sus cambios (realizados en un repositorio privado) al mismo. Para "subir" sus cambios al repositorio canónico cada desarrollador tiene que hacer un pedido de fusión (*merge request*) a la persona responsable del mismo.
- La principal ventaja de esta forma de trabajar es que se puede continuar trabjando, y el responsable del repositorio canónico puede integrar los cambios en cualquier momento. Cada cual puede trabajar a su propio ritmo.

Flujo de trabajo con Dictador y Tenientes

- Es una ampliación del flujo de trabajo con gestor de integraciones. Es utilizado en proyectos muy grandes, con cientos de colaboradores, como el kernel de Linux.
- Se distribuye la gestión de integración en partes concretas del repositorio entre los *tenientes*. Todos los tenientes rinden cuentas al *dictador*. El dictador integra todos los aportes de los tenientes publicando el trabajo en el repositorio canónico.
- Este sistema de trabajo permite al líder del grupo (el *dictador*) delegar gran parte del trabajo en los *tenientes*, relegando su trabajo en recolectar el fruto de múltiples puntos de trabajo.

> Fuente. Wikipedia: https://es.wikipedia.org/wiki/Control_de_versiones