**Buenas Prácticas de Gestión de Versiones con Subversion**

**Introducción**

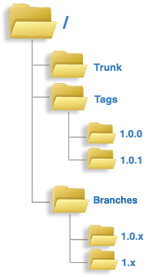
Subversion (SVN) es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros.

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco.

SVN permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado. El objetivo de este artículo es guiar al desarrollador para que sea capaz de tomar la mejor decisión en cada etapa del ciclo de vida de su proyecto.

**TTB, La Estructura Habitual Subversion**

La estructura TTB se ha convertido en el estándar de facto en los repositorios SVN. TTB son las iniciales de las tres carpetas que compondrán el primer nivel de directorios del repositorio: Trunk, Tags y Branches.



Cada carpeta tiene su funcionalidad específica, pero Subversion, al igual que un disco duro, las tratará por igual y no limitará las operaciones a realizar sobre ellas; por tanto conocer y aplicar las buenas prácticas ayudará a los usuarios a darles un uso correcto.

A continuación se listan las funcionalidades que se le debería dar a cada rama del repositorio:

* Trunk: Rama de desarrollo principal.
* Tags: Rama de gestión de versiones. Reservado para versiones cerradas (congeladas), por tanto no se desarrollará sobre esta rama.
* Branches: Rama con evoluciones paralelas al Trunk.

Los conceptos de desarrollo principal, evolución y congelación se explican a continuación.

**Operaciones Habituales con Subversion**

A continuación se presentan las operaciones más habituales con las que nos encontramos trabajando con Subversion.

**Trabajo en Equipo**

Se refiere a la situación en la que al menos dos personas modifican el código.

**Por qué:** Mientras otras herramientas obligan a bloquear zonas del repositorio cuando se estén realizando cambios en ellas, Subversion permite la modificación paralela de código del repositorio, de modo que varias personas pueden trabajar de forma simultánea sobre cualquier parte del código sin crear interferencias. En el caso de que dos desarrolladores modificasen el mismo elemento a la vez, Subversion integrará los cambios de forma automática, obligando al usuario a hacerlo de forma manual sólo en casos en los que el conocimiento humano es el único que puede asegurar la correcta integración.

**Cuándo:** Antes de hacer cualquier modificación en su entorno local, los desarrolladores deben asegurarse de estar trabajando con la última versión del software del repositorio. Lo mismo sucederá al finalizar un desarrollo: antes de persistir los cambios en el repositorio de Subversion se deberá asegurar que no se está interfiriendo con un desarrollo paralelo que ya haya sido guardado en el repositorio. Para esto se utilizará el mecanismo de sincronización de Subversion.

Buenas prácticas: Existen tres formas de sincronizar el código del entorno local con el del repositorio:

* El comando Checkout descargará al entorno local una copia fiel del código del repositorio. Útil para comenzar a desarrollar sobre proyectos nuevos.
* El comando Update descargará al entorno local únicamente las modificaciones que hayan tenido lugar desde la última sincronización. Sólo se podrá hacer esta operación si se dispone ya de una versión local del código del repositorio.
* El comando Commit actualizará el contenido del repositorio con los cambios del entorno local. Subversion sólo permitirá esta operación si no existen conflictos con el código ya existente en el repositorio. Es decir, no permitirá hacer Commit si otro miembro del equipo ha modificado el mismo elemento de forma paralela desde la última sincronización de código.

Para favorecer el trabajo en equipo, se recomienda la orientación del desarrollo a tareas de resolución a corto plazo. Aunque la gestión de las tareas se puede llevar en una hoja de cálculo o por correo electrónico, también existen herramientas específicas para la gestión de tareas en proyectos de desarrollo de software como Atlassian Jira, o Bugzilla. El funcionamiento de estas herramientas queda fuera del alcance de este artículo.

La dinámica habitual de trabajo deberá ser la siguiente:

1. Antes de comenzar con la resolución de una tarea, se deberá asegurar la sincronización con el repositorio, bien con un Update o bien con un Checkout dependiendo de si se dispone previamente del código en el entorno local o no.
2. Una vez resuelta la tarea, se deberá hacer otro Update para traer al entorno local los cambios que hayan podido ser realizados en paralelo al desarrollo actual. Subversion sabrá integrar los cambios del repositorio con los del entorno local en la mayoría de los casos, pero existirán situaciones que requieran de intervención humana para la integración. Estos casos, se deberán resolver de forma manual procurando mantener las modificaciones propias y las realizadas por los otros desarrolladores en paralelo.
3. Finalmente se deberá hacer el Commit para hacer público al resto del equipo el código desarrollado. El alcance del Commit deberá limitarse al código relevante a la resolución de la tarea, y no mezclar desarrollos de distintas tareas en un mismo Commit.

**Cierre de Versión (Creación de Tag)**

**Por qué:** En ciertos momentos del ciclo de vida de un proyecto de software puede ser conveniente el cierre de una versión para continuar con su evolución en el ámbito de la versión siguiente. Este cierre de versión nos permitirá volver a versiones anteriores en situaciones que lo requieran. Un ejemplo puede ser la necesidad de arreglar un bug tras una entrega, donde se deberá partir de la versión entregada en lugar de la versión actual en evolución, la cual podría encontrarse en una situación inestable. A este proceso de guardar una “foto” del estado del software en un momento dado también se le conoce como “congelar una versión”.

En lenguaje Subversion, el cierre de versión se denomina crear un Tag de la versión desarrollada. Esto implica llevar una copia de la versión que se desea cerrar a la rama de gestión de versiones. Subversion maneja copias baratas para esto, es decir, sólo guarda una referencia a la rama y revisión que se desea copiar, lo que significa que el coste tanto en tiempo como en espacio en disco es bajo y constante (no es dependiente ni del número, ni del tamaño de los ficheros que componen la versión).

**Cuándo:** Dado el bajo coste de la creación de Tags para los cierres de versión, se recomienda que se realicen con cada hito del desarrollo del proyecto.

**Buenas prácticas:** Es importante no modificar nunca un Tag tras su creación, ya que se estaría perdiendo la referencia a la versión que en su momento se decidió congelar. Subversion no impide esta modificación, así que es responsabilidad de los desarrolladores el seguir esta buena práctica.

Una vez creado el Tag, se debe utilizar la rama donde se desarrolló la versión cerrada (bien el Trunk o bien un Branch) para la evolución hacia la siguiente versión.

**Ramificación del Código (Creación de Branch)**

**Por qué:** Existen situaciones en las que el ciclo de vida de un proyecto implica una evolución paralela de su código. Subversion habilita entornos disjuntos para estos desarrollos mediante la creación de Branches. Las modificaciones realizadas en los entornos paralelos pueden ser fusionadas en cualquier momento mediante la Fusión de Cambios que se explicará en el siguiente punto. Igual que en el caso de creación de Tags, Subversion también maneja copias baratas en las ramificaciones por lo que el coste es bajo.

**Cuándo:** La necesidad de bifurcar la evolución de un código puede surgir por diferentes motivos. El más habitual es la necesidad de seguir evolucionando un software al mismo tiempo que se corrigen los bugs que puedan surgir de la última versión puesta en producción. En este caso se necesitaría un branch evolutivo y otro correctivo. Otra situación puede ser la necesidad de realizar un gran número de modificaciones que durante su desarrollo obligarían a dejar el repositorio en un estado inestable, en cuyo caso se crearía un branch inestable hasta finalizar todas las modificaciones; u otras situaciones como que del proyecto surjan dos evoluciones de naturaleza distinta y por tanto no sea conveniente desarrollarlas de forma conjunta.

**Buenas prácticas:** Pese a ser un proceso de bajo coste como la creación de Tags, la ramificación del código debe realizarse sólo en casos en los que no exista alternativa a la creación de una nueva rama de desarrollo. La razón de esto es que a medida que crece el número de ramas, se hace más compleja la gestión de las versiones en desarrollo y puede llevar a los desarrolladores a perder la dirección inicial del proyecto.

Salvo en los casos en los que la ramificación de código sea con el objetivo de crear un proyecto nuevo a partir del código inicial, se deben considerar los Branches como ramas de desarrollo de vida limitada, es decir, tendrán un tiempo de vida tras el cual se deberá dejar de trabajar sobre ellos, bien por un Cierre de Versión o bien por la Fusión de Cambios a la rama de la que se hizo la ramificación.

**Fusión de Cambios**

**Por qué:** En muchos casos tras una ramificación, los cambios realizados en una rama se deben aplicar a algún desarrollo paralelo. Subversion facilita este proceso mediante el comando Merge, que aplica todos los cambios producidos entre dos revisiones en una rama a otra rama cualquiera del repositorio. En el caso de una bifurcación para la resolución de un bug en una rama correctiva de forma paralela al desarrollo de la siguiente versión en una rama evolutiva, deberán fusionarse los cambios realizados en la rama correctiva con los cambios que hayan surgido simultáneamente en la rama evolutiva.

**Cuándo:** Siempre tras la finalización de un desarrollo paralelo que afecte a alguna rama paralela.

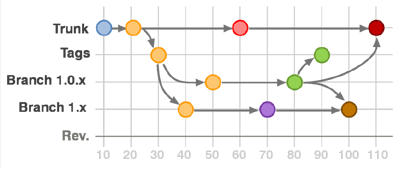
**Buenas prácticas:** Se deberá hacer uso del comando Merge, ya que la aplicación manual de los cambios es tanto susceptible de error humano como mucho más costosa en tiempo. La alternativa a Merge es la aplicación de un Patch que tendría la misma función, pero está limitada a modificaciones en ficheros, mientras que Merge tiene en cuenta modificaciones tanto de ficheros como de directorios.

**Ciclo de Vida con Subversion**

A continuación se explicará paso a paso la transición de estados de un posible software siendo desarrollado usando Subversion como herramienta de control de versiones.

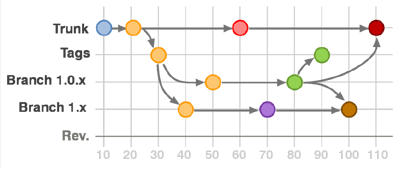
Se partirá de un código inicial que se evolucionará hasta cerrar una primera versión. Esta versión se llevará a producción y a partir de ahí se empezará a trabajar sobre la siguiente versión.

Para mostrar distintos escenarios de ramificaciones de código, se supondrá que se contrata un servicio de mantenimiento evolutivo del producto entregado, que tendrá que desarrollarse en paralelo a la evolución de la siguiente versión del producto y que además se detectará un pequeño bug en el producto que requerirá una corrección urgente, y que por tanto no podrá esperar a resolverse en una nueva versión del producto.



**Rev. 10:** Se supone un estado inicial con el código fuente de partida dado de alta en el Trunk. El resto del repositorio queda vacío.

**Rev. 20:** Evolución de la versión 1.0.0 del SW mediante Trabajo en Equipo sobre el Trunk.



**Rev. 30:** Cierre de Versión 1.0.0 del SW. Implica la creación de un Tag y el paso al desarrollo de la versión 2.0.0 en el Trunk. Suponemos la puesta en producción de la versión 1.0.0.

**Rev. 40:** Suponemos la contratación de un mantenimiento evolutivo sobre la versión puesta en producción. Esta evolución debe ser paralela al desarrollo de la versión 2.x del producto en el Trunk, por lo que necesitará una Ramificación del Código implicando la creación de un Branch para la versión 1.x.

**Rev. 50:** Suponemos la detección de un bug en la versión en producción. Para resolverlo se deberá partir del código puesto en producción, por tanto se deberá recuperar el Tag de la versión 1.0.0. Para no perder la referencia a esta versión tras la realización de cambios se deberá hacer una Ramificación del código para resolver el bug en un Branch y realizar los cambios en dicha rama, con el objetivo de crear la versión 1.0.1.

**Revs. 60–80:** Trabajo en Equipo paralelo en el Trunk para la versión 2.x, en un Branch evolutivo para la versión 1.x y en un Branch correctiva para la versión 1.0.x.

**Rev. 90:** Cierre de Versión 1.0.1 del SW. Se crea el Tag de la versión 1.0.1.

**Revs. 100-110:** Debido a que tanto el desarrollo de la versión 1.x en el Branch como el de la versión 2.x en el Trunk han partido de la versión 1.0.0 que contenía el bug resuelto en la versión 1.0.1 será altamente probable la existencia del mismo error en estas ramas. Por tanto, se deberá hacer una Fusión de Cambios de las modificaciones realizadas desde la versión 1.0.0 a la versión 1.0.1 con el estado actual del Trunk y Branch.

**Conclusión**

Se espera que el lector haya comprendido la necesidad de usar buenas prácticas al gestionar un proyecto usando una herramienta tan flexible como Subversion, y que este documento sirva de apoyo en la toma de decisiones relacionadas con el control de versiones.