LINEAMIENTOS PARA EL USO DE REPOSITORIOS DE CÓDIGO FUENTE

1. INTRODUCCIÓN

El objetivo de este documento es proporcionar lineamientos para un uso adecuado del control de versiones utilizando Subversion. Describe un proceso general que los desarrolladores pueden usar para mantener el control de su código fuente a lo largo del ciclo de vida de un proyecto. No incluye una explicación de todas las opciones y comandos de Subversion, los cuales pueden consultarse en la documentación en línea en la siguiente página Web: http://svnbook.red-bean.com/

2. REPOSITORIO Y COPIAS DE TRABAJO

Subversion almacena la historia de las versiones de archivos en un repositorio central. El repositorio central se encuentra ubicado en un servidor remoto. Cada proyecto utiliza uno o varios repositorios. A diferencia de otros sistemas de control de versiones, en Subversion no se bloquean los archivos mientras son modificados sino que cada desarrollador obtiene una copia de trabajo de una porción del repositorio en un sistema de archivos local. Después, el usuario realiza los cambios a los archivos dentro de esa copia de trabajo y finalmente los envía de nuevo al repositorio. Las nuevas versiones de los archivos se encuentran disponibles para otros usuarios.

Cada desarrollador debe tener en su equipo un área de trabajo privada, configurada con las mismas herramientas y configuración que el ambiente principal, con acceso a los elementos de configuración vigentes. De este modo, se puede tener un espacio local en el cual realizar todas las modificaciones necesarias durante el desarrollo de los componentes, sin afectar al repositorio central de código.

En el archivo de lineamientos de desarrollo de cada proyecto deben especificarse las herramientas, librerías, versiones y configuraciones requeridas para el ambiente de desarrollo local.

3. ESTRUCTURA DEL REPOSITORIO

La estructura de directorios de alto nivel de cada repositorio de código debe formarse de la siguiente manera:

trunk – directorio donde se almacena y mantiene la versión más reciente del código fuente. Los desarrolladores crean su copia de trabajo desde el trunk y ahí realizan sus modificaciones (commits).

branches – directorio que sirve para crear ramificaciones del código fuente, en donde se realizan las correcciones de defectos o se desarrollan nuevas funcionalidades para una liberación en particular. Se deben crear subdirectorios por cada rama o branch. Las ramas son útiles para trabajar líneas alternas de desarrollo de manera paralela, por ejemplo, desarrollar nuevos módulos para una futura liberación en una rama al mismo tiempo que se realizan modificaciones y liberaciones a la versión de producción, la cual se encuentra en el trunk.

tags – directorio en el que se almacenan versiones específicas o "etiquetas" del código fuente las cuales no deben modificarse. Generalmente, es utilizado para guardar las versiones liberadas o entregadas. También es útil para determinar el conjunto de archivos que contienen los cambios necesarios desde una

liberación. Se deben crear subdirectorios para cada versión o etiqueta. Por convención, los archivos que se encuentren en este directorio no deben ser editados. Si se requiere realizar modificaciones, debe crearse una rama o branch.

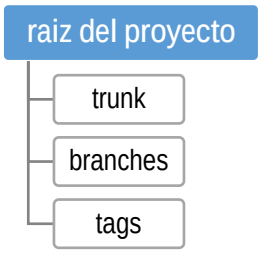


Figura 1. Estructura del repositorio

4. QUÉ SE DEBE ALMACENAR EN EL REPOSITORIO

En el repositorio se deben almacenar únicamente los archivos que permitan re-crear una liberación o entrega. Por ejemplo, en el lenguaje de programación Java, no se deben almacenar los archivos .class; únicamente se deben almacenar los archivos de código fuente como son los .java y los scripts que se utilicen para construir, ejecutar o probar el sistema.

Si resulta necesario es posible almacenar librerías externas cuando no sean fácilmente accesibles o disponibles, como es el caso de drivers especiales para conexión a base de datos.

Se deben almacenar imágenes y otros archivos que resulten necesarios para el funcionamiento del software, por ejemplo logotipos, documentos de manuales referenciados desde el sistema, entre otros.

No se deben almacenar archivos que sean utilizados para realizar pruebas o generados por el mismo sistema, por ejemplo imágenes y archivos relacionados con registros de prueba.

Tampoco se deben almacenar los archivos de configuración de herramientas locales, como por ejemplo la carpeta de metadatos que genera Netbeans (nbproject), o la carpeta de metadatos de herramientas como GoodSync (gsdata).

5. FLUJO DE TRABAJO

Para modificar el código fuente, generalmente se realizan las siguientes operaciones:

1. Se actualiza la copia de trabajo utilizando el comando `svn update`.
2. Si existen conflictos entre la copia de trabajo y el repositorio, se deben resolver y después marcar los conflictos como resueltos con `svn resolve`.
3. Se realizan los cambios en el código fuente en la copia local. Se utilizan los comandos `svn add`, `svn delete`, `svn copy` y `svn move` según se requieran.

4. Se deben probar las modificaciones realizadas, verificar el cumplimiento de estándares y corregir problemas.
    a. Se pueden revisar los cambios con apoyo de los comandos `svn status` y `svn diff`.
    b. Se pueden revertir y abandonar los cambios utilizando el comando `svn revert`.
5. Se envían los cambios al repositorio con el comando `svn commit`.

En la siguiente figura se muestra el flujo de trabajo con Subversion:

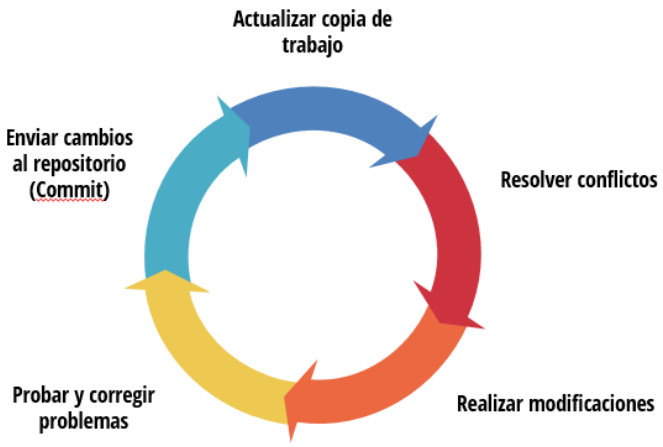


Figura 2. Flujo de trabajo típico

6. POLÍTICAS PARA EL CONTROL DE VERSIONES DEL CÓDIGO FUENTE

6.1. Probar las modificaciones antes de incorporarlas al repositorio

Antes enviar los cambios al repositorio (commit), el desarrollador debe probar que todos los scripts, compilaciones, pruebas o cualquier cosa que sea requerida en el proyecto funcionen apropiadamente. Esto con la intención de evitar problemas al resto del equipo de desarrollo.

6.2. Incorporar elementos al repositorio frecuentemente

Integrar el trabajo requiere registrar los cambios o nuevos elementos tan pronto como estén listos. De este modo, se evitan problemas de incompatibilidad o conflictos entre el código y el del resto del equipo. Además el equipo tendrá acceso oportuno a esos elementos de configuración, lo que los ayudará a su vez a integrar ese código con el suyo. Es importante subir los cambios tan frecuentemente como sea necesario, pero sólo cuando los elementos hayan pasado las verificaciones y pruebas correspondientes.

6.3. Incluir comentarios detallados en cada incorporación al repositorio (commit)

Para tener el registro de las razones de las incorporaciones y de los cambios realizados, se deben incluir comentarios detallados cuando se suban archivos al repositorio. Dentro de la información que se puede incluir, se encuentra el requerimiento o caso de uso que se incorpora, la causa original del cambio – defecto, solicitud de cambio o mejora–, el identificador del defecto o solicitud de cambios, la intención del cambio y cualquier otra información que pueda ser útil.

6.4. Cada incorporación al repositorio debe reflejar un mismo propósito

Cada commit puede incluir uno o varios archivos, pero es recomendable que los cambios realizados correspondan a una sola tarea, ya sea la corrección de un defecto, la incorporación de una mejora o nuevo requerimiento o bien la implementación de un cambio solicitado. Por ejemplo, si se realizan cambios a cinco archivos diferentes para la resolución de un defecto, debe realizarse un solo commit que incluya los cinco archivos en lugar de enviar cada archivo por separado. En cambio, si se modifican cinco archivos para solucionar cinco diferentes defectos, cada archivo se debe enviar por separado. De este modo, en caso de ser necesario, se vuelve muy sencillo analizar y revisar el cambio o bien deshacerlo para regresar a una versión anterior.

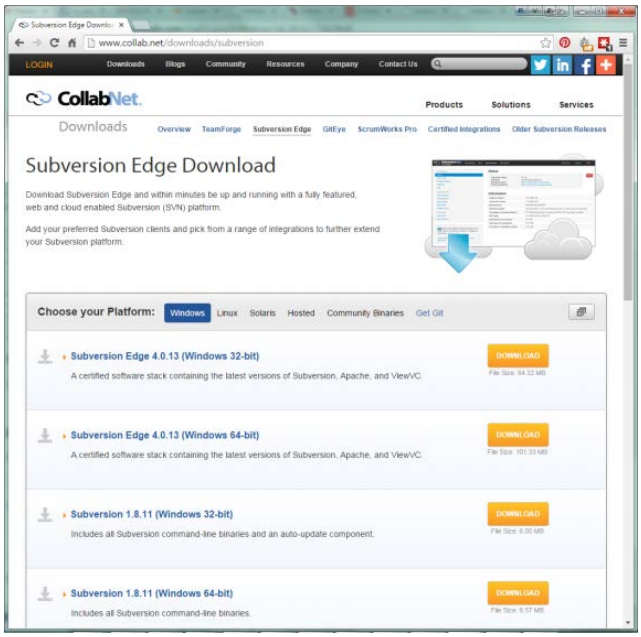
6.5. Mantener el espacio privado sincronizado con el repositorio central

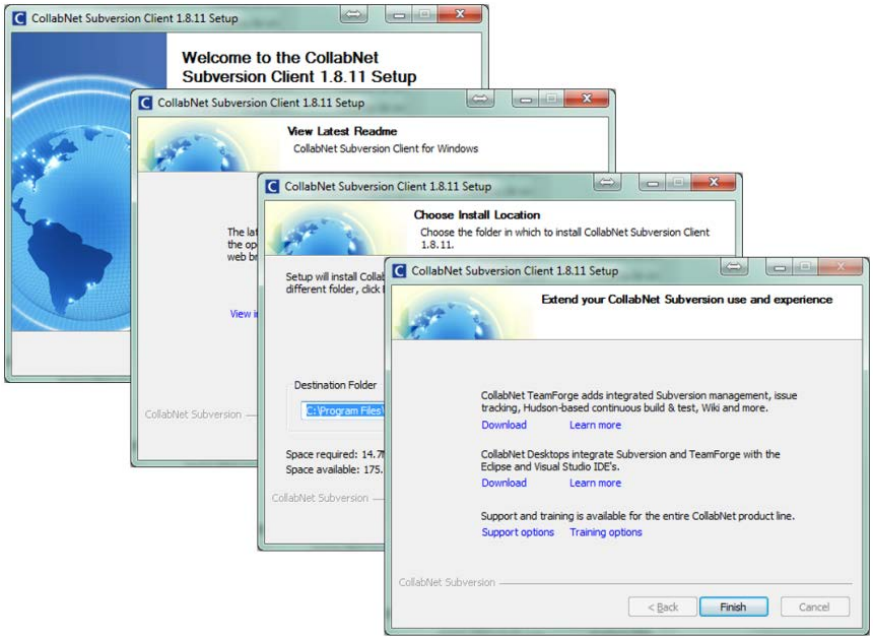
En todo momento, el desarrollador debe cuidar que su código se integre bien con el del resto del equipo. Es importante descargar continuamente las actualizaciones del repositorio central al espacio de trabajo privado para garantizar la correcta integración de los componentes en desarrollo con el resto del sistema. Especialmente, se requiere actualizar la copia local antes de enviar actualizaciones al repositorio central.

7. ANEXOS

7.1. Anexo 1: Configuración del ambiente de trabajo con Netbeans y Subversion

1. Descargar e instalar el cliente de subversión http://www.collab.net/downloads/subversion



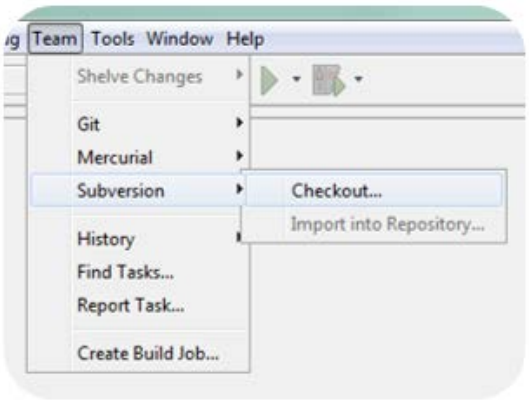


2. Obtener los datos de conexión al repositorio. Se deben solicitar al Líder Técnico.

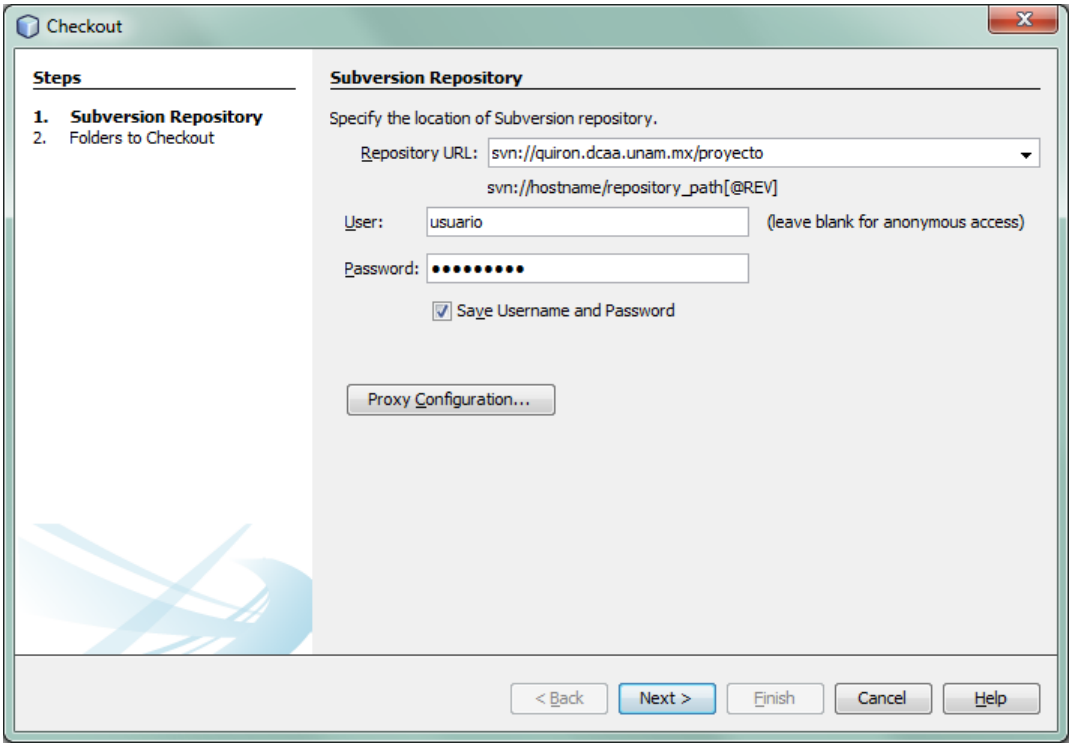
    a. URL del repositorio
    b. Usuario
    c. Contraseña

3. Sincronizar un directorio en el equipo local con el repositorio:

    a. En Netbeans, elegir la opción `Team > Subversion > Checkout` del menú principal.



    b. Se abre la ventana `Checkout`. Ingresar la URL del repositorio, usuario y contraseña.



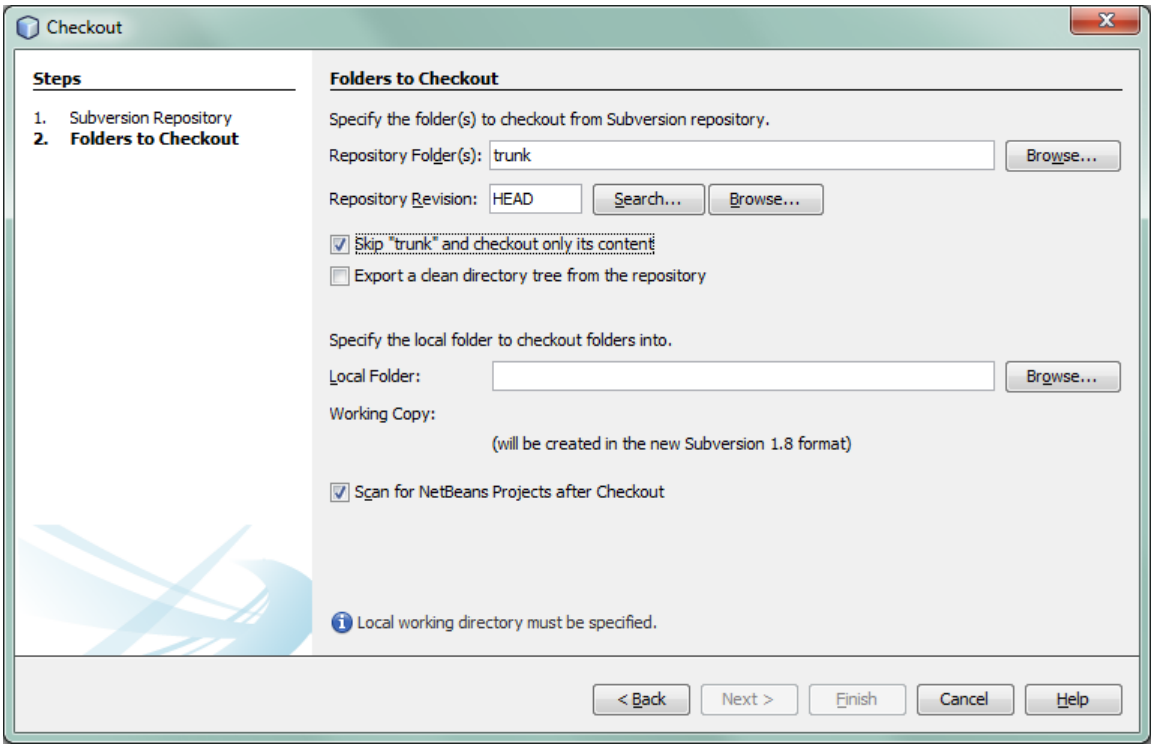
c. En la siguiente pantalla, realizar las siguientes acciones:

- Especificar la carpeta del repositorio; seleccionar la opción trunk.

- Marcar la opción Skip "trunk" and checkout only its content.

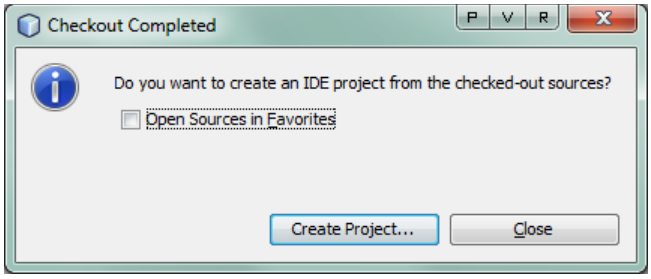
- En Local Folder, seleccionar un directorio del equipo local donde se guardará la copia de trabajo. Es importante que dicha carpeta no se encuentre dentro de un servidor, como por ejemplo htdocs, localhost, entre otros.

- Marcar la opción "Scan for Netbeans Projects after checkout

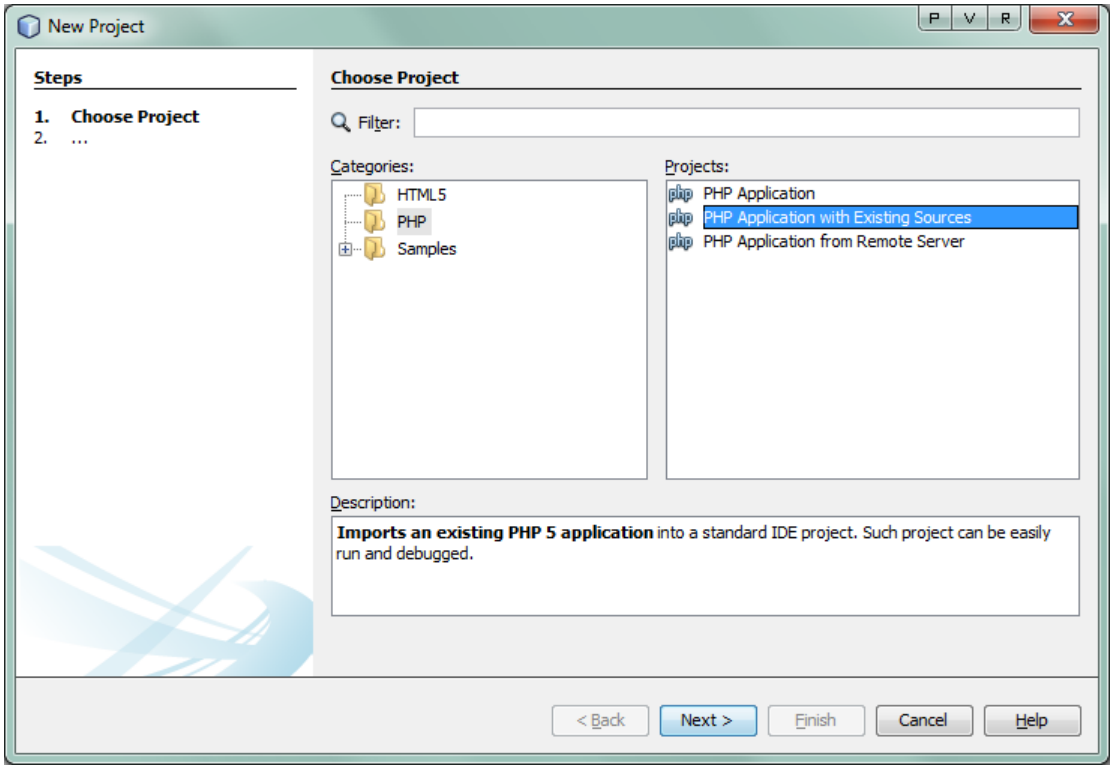


En la parte inferior de Netbeans aparecerá la barra de progreso la cual indica que se está obteniendo la copia del repositorio.

d. Al terminar el proceso, se presenta la opción para crear un proyecto en Netbeans. Dar clic en el botón `Create Project…`



e. Elegir el tipo de proyecto de acuerdo con la tecnología que se utilizará. Seleccionar la opción que indique "with Existing Sources".



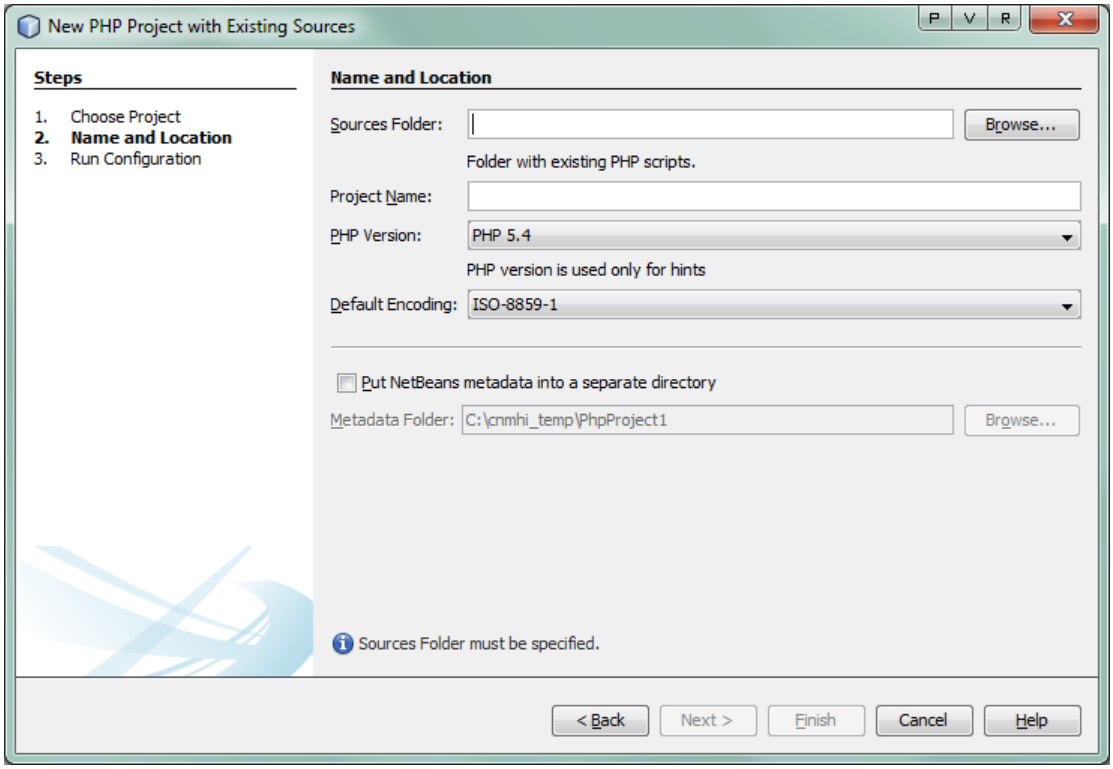
f. En la siguiente ventana ingresar:

- En Sources Folder, seleccionar el folder en donde se almacenó la copia (Local Folder del paso c).

- Ingresar un nombre del proyecto.

- Seleccionar el encoding que se utilizará en el proyecto. Consultar con el Líder Técnico.

- Marcar la opción "Put NetBeans metadata into a separate directory" y seleccionar un directorio que se encuentra fuera del directorio del proyecto. En caso de no marcar esta opción, es importante que al realizar commits al repositorio se marque como “ignorar” la carpeta *nbproject*.

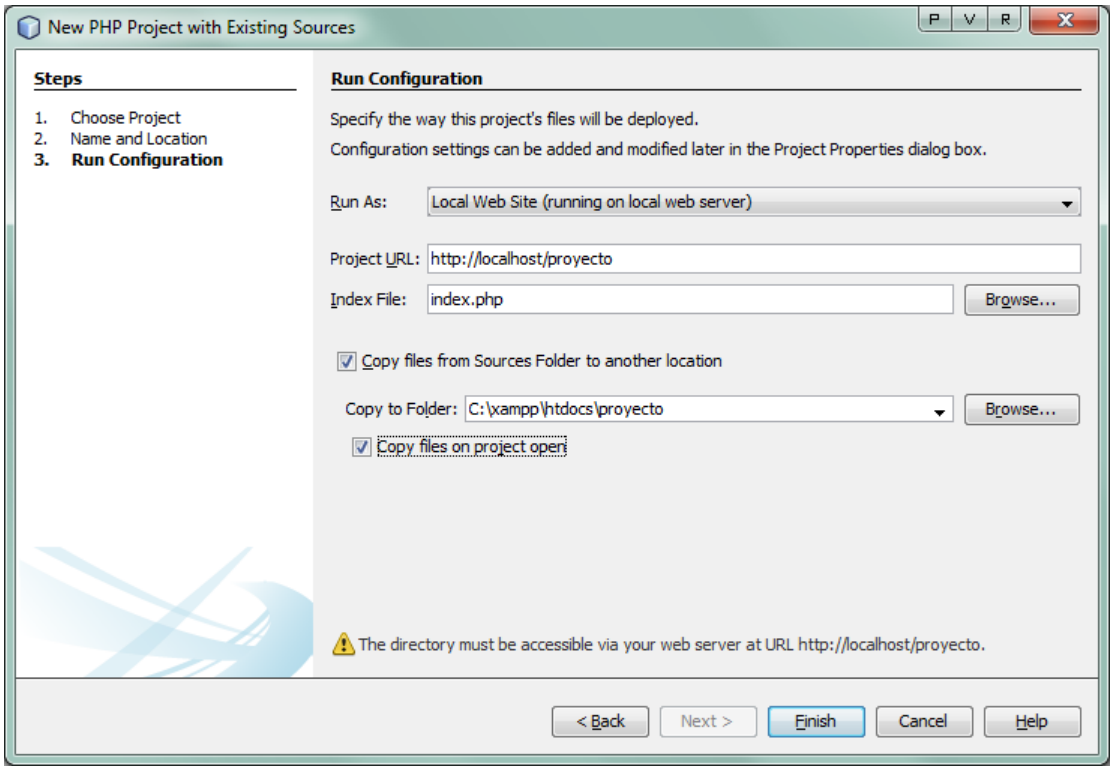


En la siguiente ventana, se configura el ambiente de ejecución, el cual puede variar dependiendo el tipo de proyecto y la tecnología a utilizar. Para proyectos PHP:

- Dejar seleccionada la opción Run As Local Web Site (running on local web server)

- Ingresar la url donde se verá el proyecto en ejecución.

- Marcar la opción Copy files from Sources Folder to another location e ingresar el folder a donde se deberá copiar el código fuente. En el caso de proyectos PHP con XAMPP, corresponde a la carpeta del proyecto dentro de C:\xammp\htdocs



Para mayor información sobre el uso de Subversion con Netbeans, consultar la siguiente dirección: https:/netbeans.org/kb/docs/ide/subversion.html

7.2. Anexo 2: Consideraciones para la creación de tags y branches

Branches

Una de las características de los sistemas de control de versiones es la posibilidad de aislar cambios en una línea separada de desarrollo. Esto se conoce como una rama o branch. Las ramas se utilizan a menudo para probar o desarrollar nuevas características sin afectar la línea principal del desarrollo (trunk) con errores de compilación o cambios en configuraciones, bases de datos, entre otros. Tan pronto como la nueva característica es lo suficiente estable, la rama de desarrollo se fusiona de nuevo en la línea principal (trunk).

Los branches se utilizan cuando se desea crear una rama independiente de desarrollo para trabajar en alguna funcionalidad nueva de un proyecto que todavía no se quiere incorporar a la línea principal. De esta forma, se crea una copia del proyecto en una carpeta branches y el programador encargado de esta nueva característica cambiará (switch, según la terminología de SVN) su working copy a este nuevo branch y trabajará sobre este.

Cuando este programador realice operaciones de commit las realizará sobre su branch, sin afectar el desarrollo principal. Cuando esté listo para integrar sus cambios a la línea de trabajo principal, realizará una operación de fusión o merge.

Tags

Otra característica de los sistemas de control de versiones es la posibilidad de marcar revisiones particulares (por ejemplo, una versión lanzada a producción), para que en cualquier momento se pueda recrear un cierto entorno o compilación. Este proceso se conoce como etiquetar (tagging).

Se deben crear tags en las siguientes situaciones:

- Cuando se realice una entrega del software al cliente, ya sea una entrega parcial o total, sin importar que el sistema vaya a ser instalado o ejecutado o no.
- Cuando ocurra algún otro hito importante en el proyecto, como cambio importante de versión, liberación a pruebas, entre otros.

Es importante etiquetar las liberaciones, ya que su uso proporciona un mecanismo simple para realizar correcciones a las entregas. Cuando se detecta un error en el código liberado, se puede crear una rama de esa versión etiquetada, implementar la corrección y liberarla. Se debe etiquetar la nueva liberación, para que en caso de que se encuentre una nueva incidencia y se requiera liberar nuevamente una corrección.

Si no se etiquetan las versiones liberadas, puede ser muy complicado obtener el código exacto que se incluyó en una liberación en particular.

Las etiquetas o tags deben considerarse como de sólo lectura, es decir, el contenido de la carpeta tags no debe modificarse por ningún motivo.

Creación de ramas y etiquetas

Técnicamente la creación de un branch y un tag es idéntica, ya que se realizan con el comando svn copy. La diferencia radica en el tratamiento o uso que se les da a cada uno: en las ramas está permitido realizar commits y en las etiquetas no. Esta "prohibición" sobre la realización de commits se da como una política de uso y acuerdo entre el equipo de desarrollo, ya que no existe impedimento técnico para la realización de modificaciones.

Subversion no tiene comandos especiales para hacer ramas (branches) o etiquetas (tags), sino que utiliza lo que se denomina "copias baratas". Las copias baratas son similares a los vínculos duros de Unix, que significa que en vez de hacer una copia completa en el repositorio, se crea un vínculo interno, apuntando a una revisión y árbol específicos. Como resultado, las ramas y las etiquetas son muy rápidas de crear, y casi no conllevan espacio extra en el repositorio.

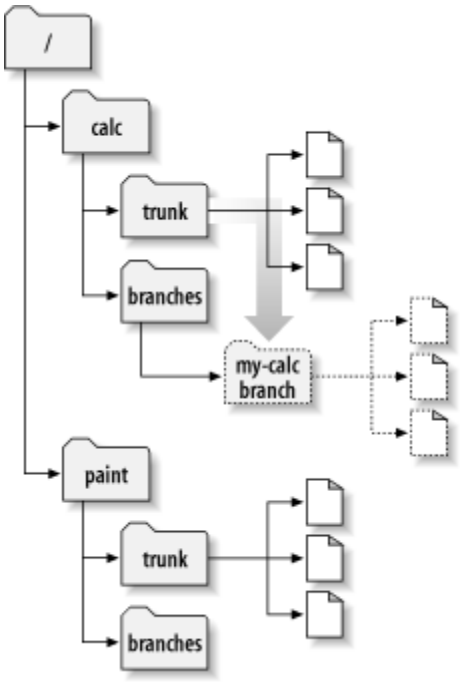
Otro punto a considerar es que las ramas deben crearse dentro del directorio /branches y las etiquetas deben crearse dentro del directorio /tags.

El comando para la creación, tanto de branches como tags, es svn copy:

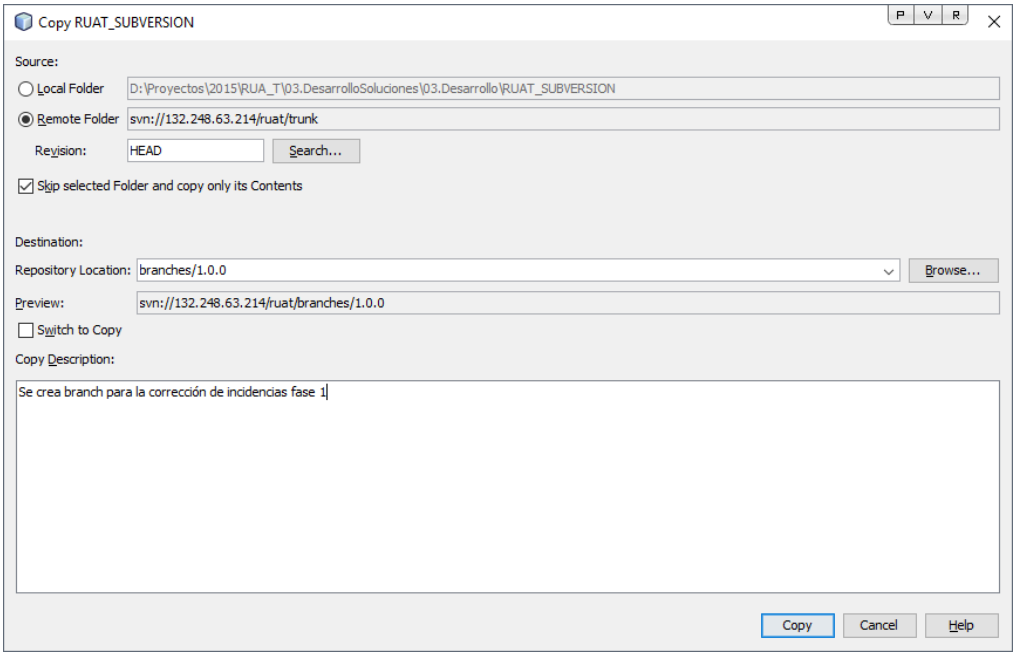
```
svn copy URL1 URL2
```

Ejemplo de creación de rama:

```
svn copy http://svn.example.com/repos/calc/trunk \
          http://svn.example.com/repos/calc/branches/my-calc-branch \
          -m "Creating a private branch of /calc/trunk."
```



Creación de branch con netbeans:


Copy RUAT_SUBVERSION
Source:
Local Folder
D:\Proyectos\2015\RUA_T\03.DesarrolloSoluciones\03.Desarrollo\RUAT_SUBVERSION
Remote Folder
svn://132.248.63.214/ruat/trunk
Revision:
HEAD
Search...
Skip selected Folder and copy only its Contents
Destination:
Repository Location:
branches/1.0.0
Browse...
Preview:
svn://132.248.63.214/ruat/branches/1.0.0
Switch to Copy
Copy Description:
Se crea branch para la corrección de incidencias fase 1
Copy
Cancel
Help

Mantener la rama sincronizada con el trunk

Al trabajar con ramas, es importante mantenerlas actualizadas respecto de los cambios que se puedan seguir desarrollando en el trunk. Retrasar la sincronización o fusión entre ambas partes, puede ocasionar múltiples conflictos difíciles de resolver.

Mientras que las ramas se utilizan para mantener líneas de desarrollo separadas, en alguna etapa se tendrán que fusionar los cambios hechos en una rama de vuelta en el tronco, o viceversa.

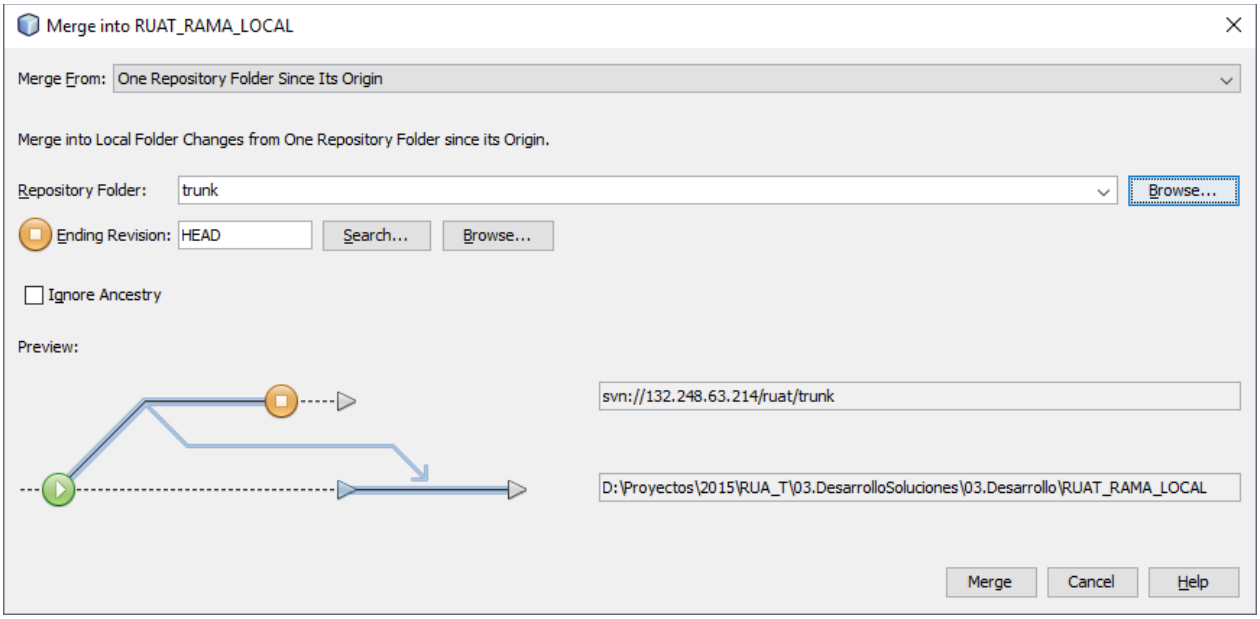
En general es una buena idea realizar fusiones en una copia de trabajo sin modificar y actualizada. Si ha hecho otros cambios en su copia de trabajo, confírmelos primero (commit).

Mantener frecuentemente la rama en sincronía con el trunk ayuda a prevenir conflictos inesperados cuando sea necesario reintegrar los cambios al trunk.

Para realizar la fusión, primero necesitamos asegurarnos que la copia de trabajo del branch no tiene modificaciones (se han realizado todos los commit necesarios) y se encuentra actualizada. Después, ubicados en el directorio de la copia de trabajo del branch, se ejecutan el comando:

svn merge ^/calc/trunk

O bien, utilizando Netbeans:



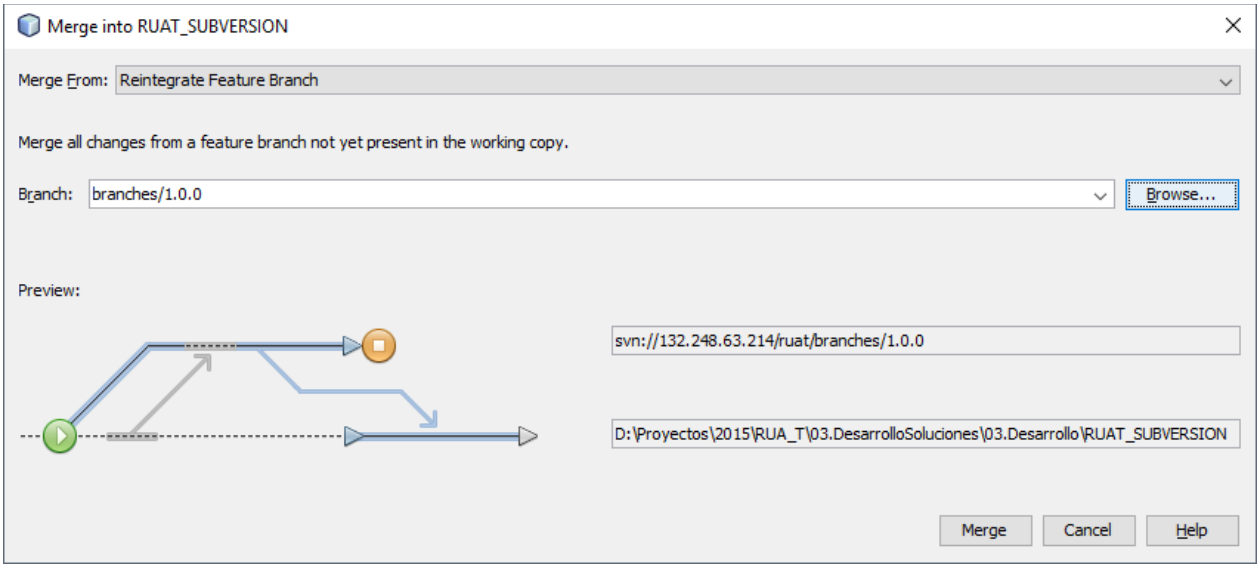
Debido a que la fusión siempre tiene lugar en una copia de trabajo, después de realizar la fusión o merge será necesario confirmar las actualizaciones al repositorio realizando un commit. En este caso es imprescindible que el comentario de la confirmación indique que corresponde a una fusión.

Reintegrar la rama al trunk

Cuando se ha concluido el desarrollo o correcciones en una rama, el siguiente paso es reintegrar esos cambios al trunk, para que estén disponibles en la línea principal de desarrollo.

Para ello,

1. Actualizar la copia de trabajo local de la rama (update)
2. Incorporar todas las modificaciones al repositorio de la rama (commit)
3. Cambiarse a la copia de trabajo local del trunk, ya sea utilizando el comando switch o haciendo un checkout del trunk. La copia de trabajo local debe estar actualizada.
4. Ubicado en el directorio de la copia de trabajo del trunk, ejecutar el comando svn merge indicando como parámetro el branch que se desea integrar. O bien, desde Netbeans:



Para más información sobre el uso de branches y tags en Subversion consulta: http://svnbook.red-bean.com/en/1.7/svn-book.html#svn.branchmerge