SVN Document

Document ID: DRSYS-SVN-01

Author(s): Balakrishnan.A, Rajesh.K

Version: 1.0

Date: Sep 08, 2014

Table of Contents

[1. Abstract 3](#_Toc398045013)

[2. Introduction 3](#_Toc398045014)

[3. Repository Creation with Tortoise SVN 3](#_Toc398045015)

[4. Repository Backup 4](#_Toc398045016)

[5. Local Access to the Repository 4](#_Toc398045017)

[6. Importing the Project into Repository 5](#_Toc398045018)

[7. Checking out a working Copy 6](#_Toc398045019)

[8. Making Changes 8](#_Toc398045020)

[9. Committing Changes 9](#_Toc398045021)

[10. Revert the Changes…………………………………………………...10](#_Toc398045022)

[11. Locking………………………………………………………………...11](#_Toc398045023)

[12. SVN with RubyMine………………………………………………….14](#_Toc398045024)

[12.1 Configure Subversion Repository Location………………………..14](#_Toc398045025)

[12.2 Importing Directory to Subversion Repository……………………15](#_Toc398045026)

[12.3 Exporting Information from Subversion Repository……………..17](#_Toc398045027)

[12.4 Checking out files from Subversion Repository……….…………..17](#_Toc398045028)

[12.5 Locking and Unlocking Files and Folders………………………….20](#_Toc398045029)

[13. Modification history…………………………………………………..21](#_Toc398045030)

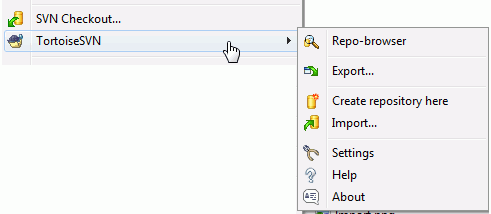
# Abstract

The purpose of the document is to describe the how to use the SVN for our Ruby based Test Automation Project.

# Introduction

While developing scripts for an automation project, it is good to keep the back up for code changes. This will help everyone to understand the purpose of the script lines and changes created by all the team members. This will increase productivity, reduce rework efforts as well as easy to analyze.

# Repository Creation with Tortoise SVN



Steps for creating the repository

1. Open the windows explorer
2. Create a new folder and name it e.g. SVNRepository
3. Right click on the newly created folder and select TortoiseSVN → Create Repository here....
4. A repository is then created inside the new folder. Don't edit those files yourself! If you get any errors make sure that the folder is empty and not write protected.
5. You will also be asked whether you want to create a directory structure within the repository.
6. Tortoise SVN will set a custom folder icon when it creates a repository so you can identify local repositories more easily. If you create a repository using the official command line client this folder icon is not assigned.

# Repository Backup

Whichever type of repository you use, it is vitally important that you maintain regular backups, and that you verify the backup. If the server fails, you may be able to access a recent version of your files, but without the repository all your history is lost forever.

The recommended method is to run

svnadmin hotcopy Repository/Path BackUp/Path --clean-logs

to create a copy of your repository in a safe manner. Then backup the copy. The --clean-logs option is not required, but removes any redundant log files when you backup a BDB repository, which may save some space.

The svnadmin tool is installed automatically when you install the Subversion command line client. The easiest way to get this is to check the option to include the command line tools when installing TortoiseSVN, but if you prefer you can download the latest version of command line tools directly from the [Subversion](http://subversion.apache.org/packages.html#windows)website.

# Local Access to the Repository

To access your local repository you need the path to that folder. Just remember that Subversion expects all repository paths in the form file:///C:/SVNRepository/. Note the use of forward slashes throughout.

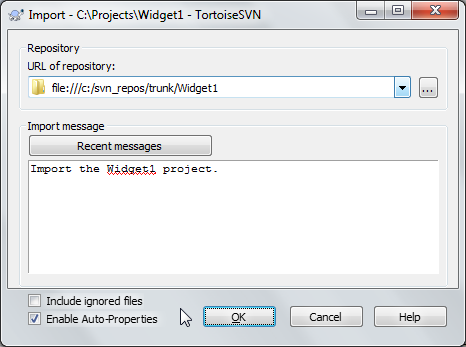
To access a repository located on a network share you can either use drive mapping, or you can use the UNC path. For UNC paths, the form is file://ServerName/path/to/repos/. Note that there are only 2 leading slashes here.

Prior to SVN 1.2, UNC paths had to be given in the more obscure form file:///\ServerName/path/to/repos. This form is still supported, but not recommended.

# Importing the Project into Repository

Now we have a repository , but it is completely empty at the moment. Let’s assume I have a set of files in C:\Projects\Widget1 that I would like to add. Navigate to widget1 folder in explorer and right click on it. Now select TortoiseSVN 🡪 Import which bring up a dialog

**Figure 1.2. The Import dialog**



A Subversion repository is referred to by URL, which allows us to specify a repository anywhere on the Internet. In this case we need to point to our own local repository which has a URL of file:///c:/svn\_repos/trunk, and to which we add our own project name Widget1. Note that there are 3 slashes after file: and that forward slashes are used throughout.

The other important feature of this dialog is the Import Message box which allows you to enter a message describing what you are doing. When you come to look through your project history, these commit messages are a valuable guide to what changes have been made and why. In this case we can say something simple like “Import the Widget1 project”. Click on OK and the folder is added to your repository.

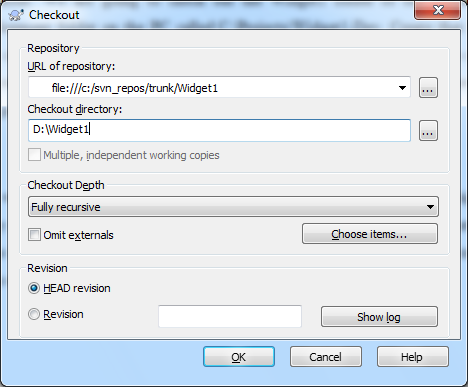
**Import in Place**

Assuming you already have a repository, and you want to add a new folder structure to it, just follow these steps:

1. Use the repository browser to create a new project folder directly in the repository. If you are using one of the standard layouts you will probably want to create this as a sub-folder of trunk rather than in the repository root. The repository browser shows the repository structure just like Windows explorer, so you can see how things are organised.
2. Checkout the new folder over the top of the folder you want to import. You will get a warning that the local folder is not empty. Ignore the warning. Now you have a versioned top level folder with unversioned content.
3. Use TortoiseSVN → Add... on this versioned folder to add some or all of the content. You can add and remove files, set svn:ignore properties on folders and make any other changes you need to.
4. Commit the top level folder, and you have a new versioned tree, and a local working copy, created from your existing folder.

# Checking out a working Copy

Now that we have a project in our repository, we need to create a working copy to use for day-to-day work. Note that the act of importing a folder does not automatically turn that folder into a working copy. The Subversion term for creating a fresh working copy is Checkout. We are going to check out the Widget1 folder of our repository into a development folder on the PC called C:\Projects\Widget1-Dev. Create that folder, then right click on it and select Tortoise SVN → Checkout.... Enter the URL to checkout, in this case file:///c:/svn\_repos/trunk/Widget1 and click on OK. Our development folder is then populated with files from the repository.



You will notice that the appearance of this folder is different from our original folder. Every file has a green check mark in the bottom left corner. These are Tortoise SVN's status icons which are only present in a working copy. The green state indicates that the file is unchanged from the version in the repository.

**Checkout Depth**

You can choose the *depth* you want to checkout, which allows you to specify the depth of recursion into child folders. If you want just a few sections of a large tree, You can check out the top level folder only, then update selected folders recursively.

**Fully recursive**

Check out the entire tree, including all child folders and sub-folders.

**Immediate children, including folders**

Checkout the specified directory, including all files and child folders, but do not populate the child folders.

**Only file children**

Check out the specified directory, including all files but do not checkout any child folders.

**Only this item**

Check out the directory only. Do not populate it with files or child folders.

**Working copy**

Retain the depth specified in the working copy. This option is not used in the checkout dialog, but it is the default in all other dialogs which have a depth setting.

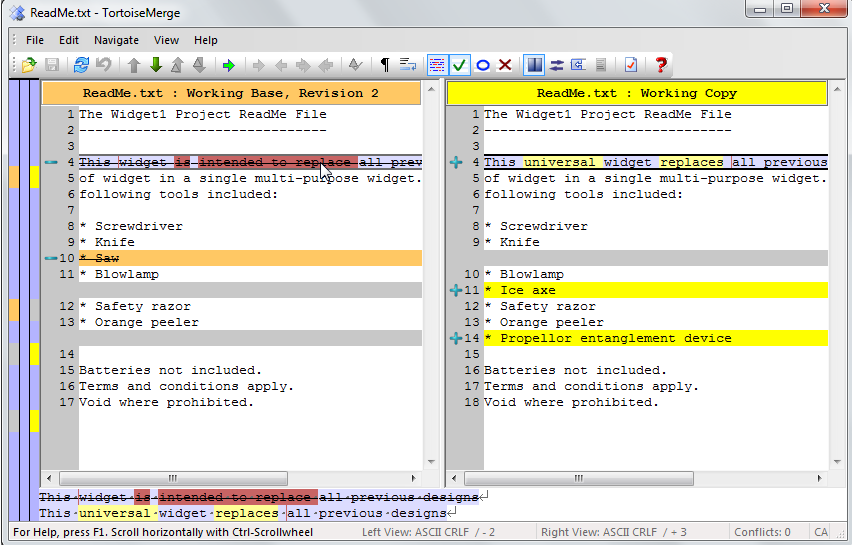
**Exclude**

Used to reduce working copy depth after a folder has already populated. This option is only available in the Update to revision dialog.

# Making Changes

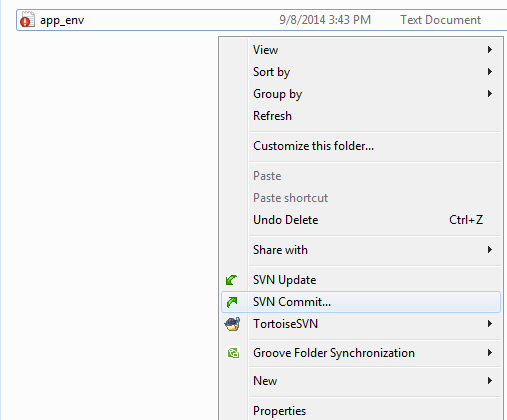
Time to get to work. In the Widget1-Dev we start editing files - let's say we make changes to Widget1.c and ReadMe.txt. Notice that the icon overlays on these files have now changed to red, indicating that changes have been made locally.

But what are the changes? Right click on one of the changed files and select TortoiseSVN → Diff. TortoiseSVN's file compare tool starts, showing you exactly which lines have changed.



# Committing Changes

Once completing the changes, let's update the repository. This action is referred to as a Commit of the changes. Right click on the Widget1-Dev folder and select TortoiseSVN → Commit. The commit dialog lists the changed files, each with a checkbox. You might want to choose only a subset of those files, but in this case we are going to commit the changes to both files. Enter up a message to describe what the change is all about and click on OK. The progress dialog shows the files being uploaded to the repository and you're done.



# Revert the Changes

One feature of all revision control systems is that they let you undo changes that you made previously. As you would expect, TortoiseSVN makes this easy to access.

If you want to get rid of changes that you have not yet committed and reset your file to the way it was before you started editing, TortoiseSVN → Revert is your friend. This discards your changes (to the Recycle bin, just in case) and reverts to the committed version you started with. If you want to get rid of just some of the changes, you can use TortoiseMerge to view the differences and selectively revert changed lines.

# Locking

**Meaning of Lock:**

Words “lock” and “locking” describe a mechanism for mutual exclusion between users to avoid clashing commits. Unfortunately, there are two other sorts of “lock” with which Subversion, and therefore this book, sometimes needs to be concerned.

The second is working copy locks, used internally by Subversion to prevent clashes between multiple Subversion clients operating on the same working copy. Usually you get these locks whenever a command like update/commit/... is interrupted due to an error. These locks can be removed by running the cleanup command on the working copy, as described in [the section called “Cleanup”](http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-cleanup.html).

And third, files and folders can get locked if they're in use by another process, for example if you have a word document opened in Word, that file is locked and cannot be accessed by Tortoise SVN.

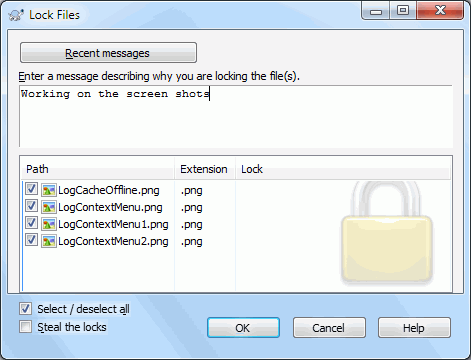
**Locks works in Sub Version:**

A lock is not assigned to a specific user, but to a specific user and a working copy. Having a lock in one working copy also prevents the same user from committing the locked file from another working copy.

As an example, imagine that user Jon has a working copy on his office PC. There he starts working on an image, and therefore acquires a lock on that file. When he leaves his office he's not finished yet with that file, so he doesn't release that lock. Back at home Jon also has a working copy and decides to work a little more on the project. But he can't modify or commit that same image file, because the lock for that file resides in his working copy in the office.

**Getting the lock:**

Select the file(s) in your working copy for which you want to acquire a lock, and then select the command Tortoise SVN → Get Lock....



A dialog appears, allowing you to enter a comment, so others can see why you have locked the file. The comment is optional and currently only used with SVN server based repositories. If (and *only* if) you need to steal the lock from someone else, check the Steal lock box, then click on OK.

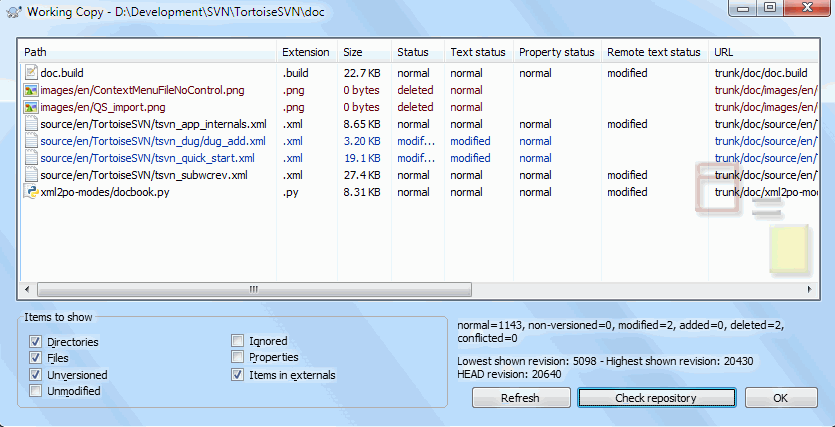
If you select a folder and then use TortoiseSVN → Get Lock... the lock dialog will open with *every* file in *every* sub-folder selected for locking. If you really want to lock an entire hierarchy then that will not make any sense.

**Releasing a Lock:**

To make sure you don't forget to release a lock you don't need any more, locked files are shown in the commit dialog and selected by default. If you continue with the commit, locks you hold on the selected files are removed, even if the files haven't been modified. If you don't want to release a lock on certain files, you can uncheck them (if they're not modified). If you want to keep a lock on a file you've modified, you have to enable the Keep locks checkbox before you commit your changes.

To release a lock manually, select the file(s) in your working copy for which you want to release the lock, then select the commandTortoiseSVN → Release Lock There is nothing further to enter so TortoiseSVN will contact the repository and release the locks. You can also use this command on a folder to release all locks recursively.

**Checking Lock status:**



To see what locks you and others hold, you can use TortoiseSVN → Check for Modifications.... Locally held lock tokens show up immediately. To check for locks held by others (and to see if any of your locks are broken or stolen) you need to click on Check Repository.

**Discovering Lock:**

Whenever you going to check out the files or folder, the best practice is to verify the lock details of the respective files.

The steps to discovering the lock details of the files or folder

1. Open the command prompt
2. Execute the status info query.

EX: SVN info “file://aspire646/SVN/RepoPath/trunk/FileorFolder”

# SVN with RubyMine

# Configure Subversion Repository Location

To configure a subversion repository location , we have to follow the below steps

1. Open SVN Repositories tool window. You can do that in a number of ways, for example:

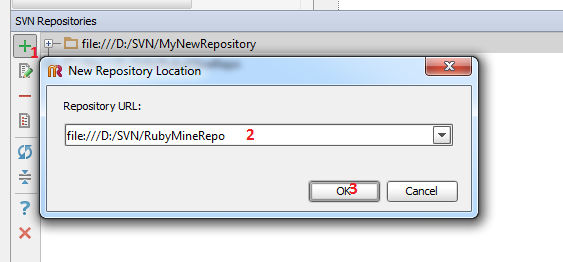
* Browse repository using VCS | Browse Subversion Repository command.
* Show the tool window using View | Tool Windows | SVN Repositories command.

1. In the SVN Repositories tool window, do one of the following:

* On the context menu of the window, choose New | Repository Location.
* On the toolbar, click add button.

You can also configure repository location from the [Import into Subversion](about:blanktopicId1014.html), or from [Check Out from Subversion](about:blanktopicId2605.html) dialog box. Use the add button, which is available on the toolbar of the dialog box.

1. In the New Repository Location dialog box, specify URL of the repository, and click OK.



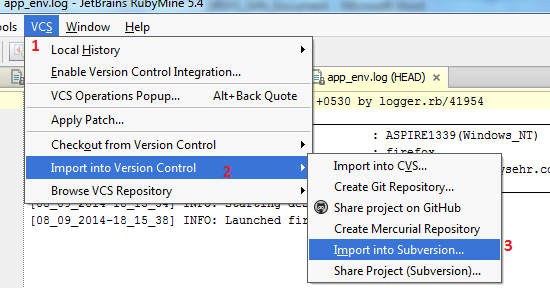
# Importing Directory to Subversion Repository

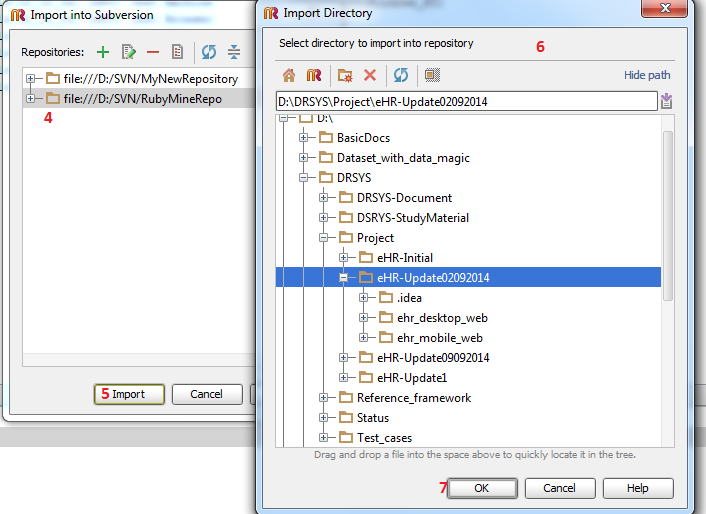
To import a directory into a subversion repository, we have to follow the below steps

1. On the main menu, choose VCS | Import into Subversion.
2. In the [Import into Subversion](about:blanktopicId303356.html) dialog box, select the target Subversion repository location. If the desired target repository location does not exist, you can [configure a new one](about:blanktopicId1008.html).
3. Click Import.
4. This action is also available in the SVN Repositories browser. Right-click the target directory and choose the command on the context menu.
5. In the Select Path dialog box, specify the directory to be imported, and click OK.
6. In the SVN Import Options dialog box, showing the read-only target repository location, and an editable field with the source directory, specify the following import options:

* Check the option Import directories recursively, if you want to upload the nested directories to the repository location.
* Check the option Include ignored resources, if you want to upload files from the ignored list.
* Enter your comment in the Commit Message field, or select one from the history drop-down list.

1. Click OK.





# Exporting Information from Subversion Repository

You might need to obtain a clean local copy of the Subversion working tree without .svn catalogs. Instead of checking files out, and then manually deleting administrative directories, you can use Export command, which is available in the Subversion repository browser.

To export a directory from subversion repository

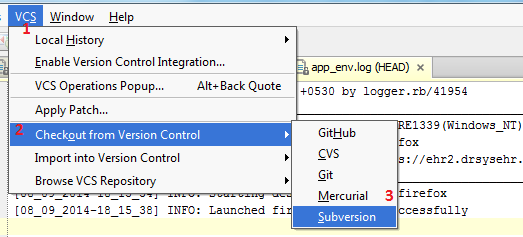
1. [Open SVN Repositories browser](about:blanktopicId994.html).
2. Right-click a directory to be exported, and choose Export on the context menu.
3. In the Select Path dialog specify the destination directory, and click OK.
4. In the SVN Export Options dialog box, specify the following options that directly map to the options of SVN export command:
5. Export recursively
   * Replace existing files
   * Include externals
   * Override native EOLs

# Checking out files from Subversion Repository

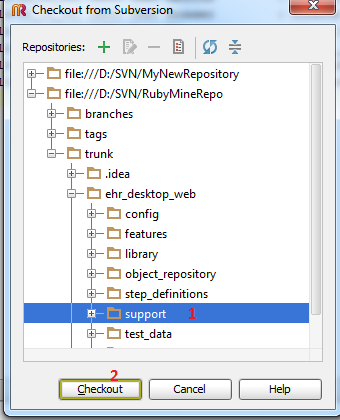
Checkout helps you obtain a [local working copy of the repository](about:blanktopicId1003.html), which you can edit as required. After making the necessary changes, you can publish results by [committing, or checking in](about:blanktopicId919.html) changes to the repository.

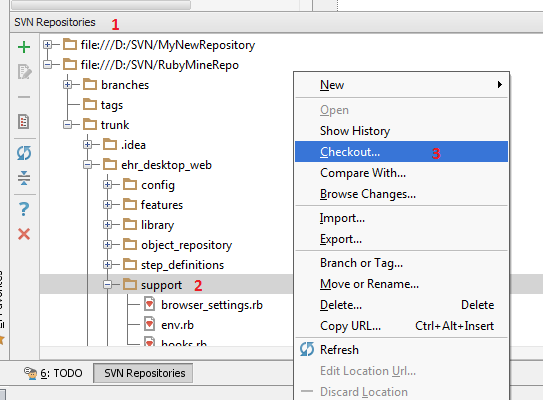
To obtain a local working copy of a subversion repository, we have to follow the following steps

1. On the main menu, choose VCS | Checkout from Version Control.
2. On the submenu, choose Subversion.



1. In the [Check out from Subversion dialog box](about:blanktopicId2605.html), expand the desired repository location and select the element to be checked out.
2. Click Checkout button.



This action is also available in the SVN Repositories browser. Right-click the desired directory and choose the command on the context menu.

1. In the [dialog that opens](about:blanktopicId580215.html), specify the destination directory where the local copy of the repository files will be created and click OK.

* If you are checking out sources for an existing project, the destination folder should be below a project [content root](about:blanktopicId871917.html).

1. In the [SVN Checkout Options](about:blankSVN_Checkout_Options_Dialog.html) dialog box, specify the following settings:

* Revision to be checked out (HEAD or a selected revision).
* Whether you need to check out the nested directories.
* Whether you need to include the external locations.

Click OK.

1. RubyMine suggests creating a project based on the sources, checked out from version control.

* If you accept the suggestion, open new project, as described in the section [Opening Multiple Projects](about:blanktopicId344.html).

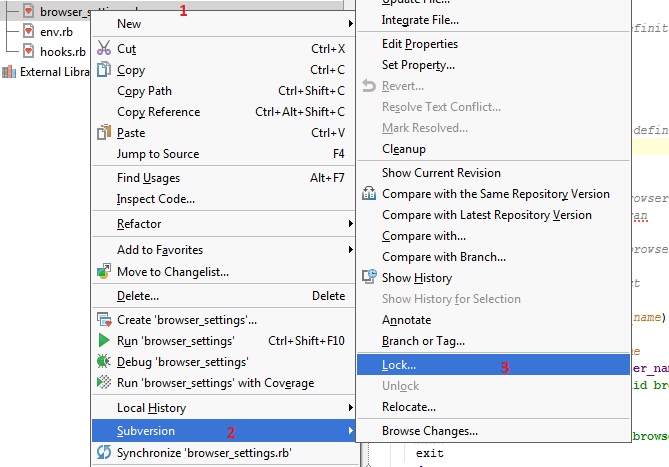
# Locking and Unlocking Files and Folders

Though Subversion integration enables you to successfully modify and merge files, changed by different team members, sometimes it makes sense to lock files (for example, images) to avoid overwriting changes.

**To lock a file**

1. Select the desired file or open it in the editor.
2. Choose VCS | Subversion | Lock on the main menu or Subversion | Lock on the context menu of the selection.
3. In the [Lock File](about:blanktopicId69027.html)  dialog box that opens specify the lock name.
4. To forcibly break the lock set by somebody else, select the Steal Existing Lock check box.

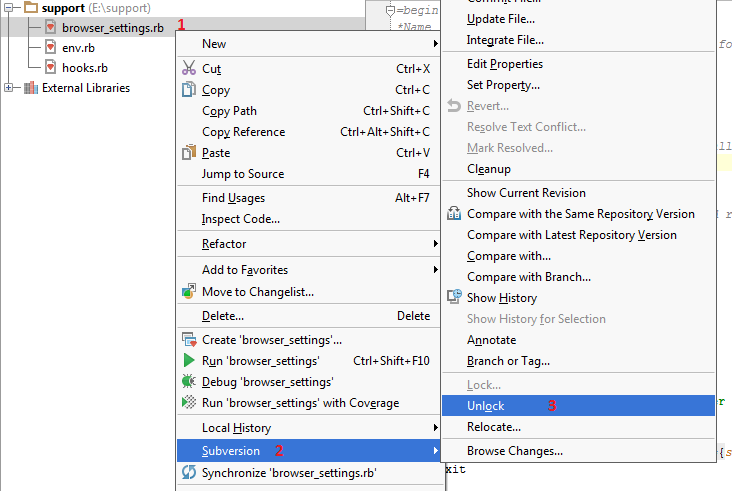
Note / Pre Condition: performing the lock, check out file should be already done.



**To remove lock from a file**

1. Select the desired file or open it in the editor.
2. Choose VCS | Subversion | Unlock on the main menu or Subversion | Unlock on the context menu of the selection.

Note / Pre Condition: To perform unlock, check out file should be already done.



# Modification history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author(s) | Modification History | Version |
| 09 Sep 2014 | Balakrishnan A | Initial document draft | 0.1 |
|  | Rajesh K | Reviewed and modified the documents |  |