GAMELOFT VIETNAM

# Tortoise SVN

DAD Studio, 2018

# Content

# Version Control and Version Control System

## Version control

It is also known as Revision Control or Source control, is the management of changes to documents,computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number," "revision level," or simply "revision."

For example, an initial set of files is "revision 1." When the first change is made, the resulting set is "revision 2," and so on.

Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

## Version control system

It is also known as a Revision Control System, is software which 'server' that practice.

**Why use it?**

- To keep multiple versions of a set of information
- Who did what and when?
- Easy to 'roll back' to an earlier version
- Reduce duplication of work
- Status of a project is always updated to every member of a team

# Tortoise SVN

## Introduction

TortoiseSVN is a really easy to use Revision control / version control / source control software for Windows. It is based on Apache™ Subversion (SVN)®; TortoiseSVN provides a nice and easy user interface for Subversion.

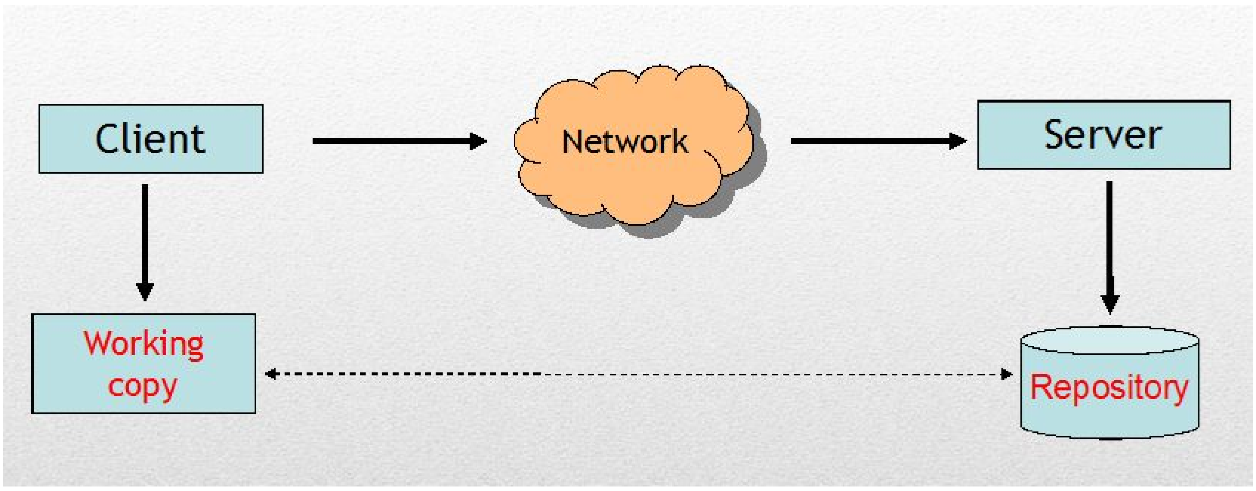
It is developed under the GPL. Which means it is completely free for anyone to use, including in a commercial environment, without any restriction.

## Download and Installing TortoiseSVN

Download new version in http://tortoisesvn.net/downloads.html

TortoiseSVN comes with an easy to use installer. Double click on the installer file and follow the instructions. The installer will take care of the rest. Don't forget to reboot after installation.

# Basic concepts



## The Repository

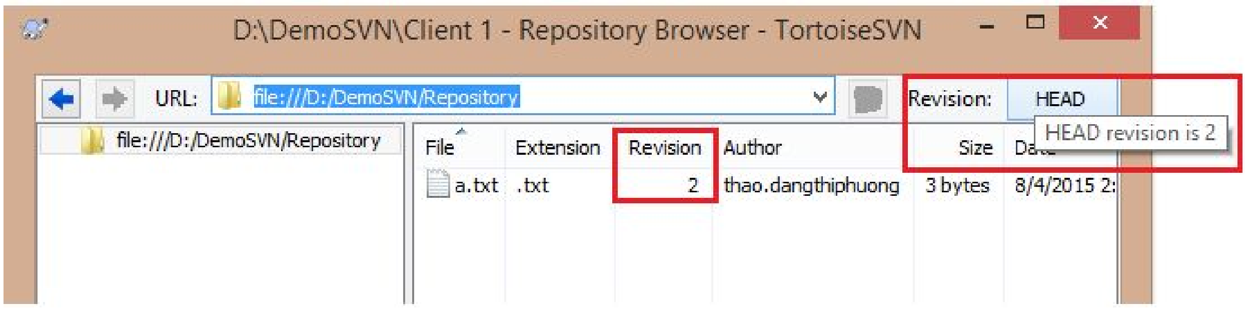
Subversion uses a central database which contains all your version-controlled files with their complete history. This database is referred to as the *repository*. The repository normally lives on a file server running the Subversion server program, which supplies content to Subversion clients (like TortoiseSVN) on request. If you only backup one thing, back up your repository as it is the definitive master copy of all your data.

## Working Copy

This is where you do the real work. Every developer has his own working copy, sometimes known as a sandbox, on his local PC. You can pull down the latest version from the repository, work on it locally without affecting anyone else, then when you are happy with the changes you made commit them back to the repository.
A Subversion working copy does not contain the history of the project, but it does keep a copy of the files as they exist in the repository before you started making changes. This means that it is easy to check exactly what changes you have made.

## Revision

Revision is  a version of a file or a group of files.
**HEAD**: The latest revision of a file or folder in the repository.


D:\DemoSVN\Client 1 - Repository Browser - TortoiseSVN
URL: file:///D:/DemoSVN/Repository
Revision: HEAD
HEAD revision is 2
file:///D:/DemoSVN/Repository

| File | Extension | Revision | Author | Size | Da... |
|---|---|---|---|---|---|
| a.txt | .txt | 2 | thao.dangthiphuong | 3 bytes | 8/4/2015 2: |

**BASE**: The “pristine” revision of an item in a working copy (or, to put it simply: it’s the latest unmodified version).

## Repository URLs

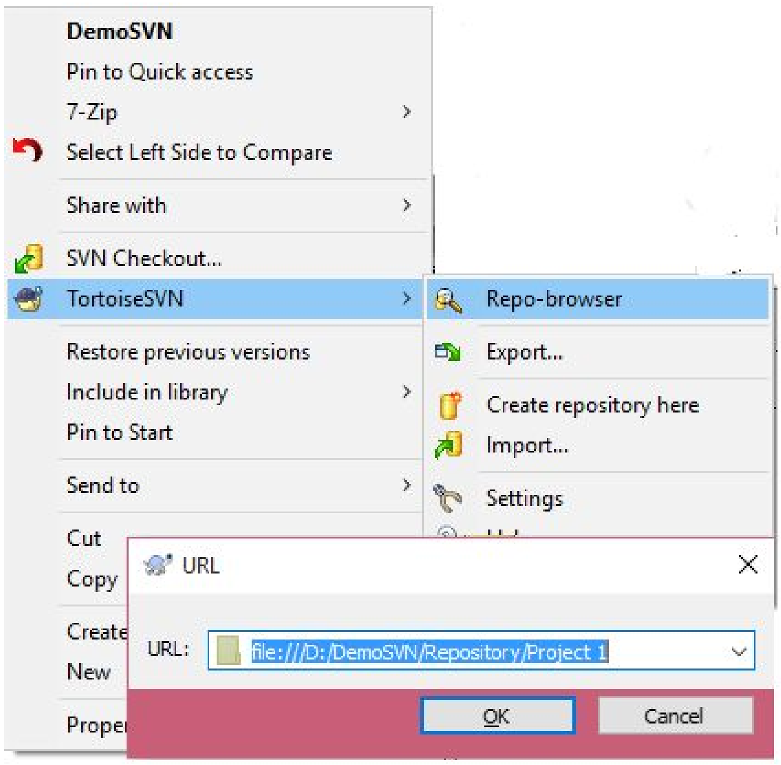
Subversion repositories can be accessed through many different methods - on local disk, or through various network protocols. A repository location, however, is always a URL. The URL schema indicates the access method:

| Schema | Access Method |
|---|---|
| file:// | Direct repository access on local or network drive. |
| http:// | Access via WebDAV protoccl to Subversion-aware Apache server |
| https:// | The same as http://, but with SSL encryption |
| svn:// | Unauthenticated TCP/TP access via custom protocol to svn server |
| svn+ssh:// | authenticated, encrypted TCP/IP access via custom protocol to a svnserve server |

## Access to the Repository

### Local server

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:/DemoSVN/Repository*. Note the use of forward slashes throughout.



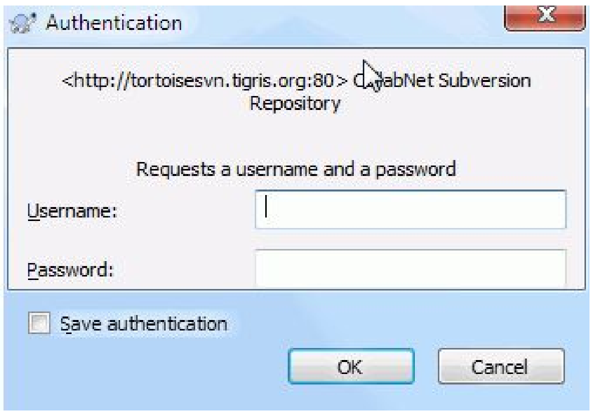
URL: file:///D:/DemoSVN/Repository/Project 1

### Gameloft server

To access a SVN Link on Gameloft server you need to request access permission it. In each studio, we have SVN team that will make this for you.

## Authentication

If the repository that you are trying to access is password protected, an authentication Dialog will show up.



Authentication

<http://tortoisesvn.tigris.org:80> CollabNet Subversion Repository

Requests a username and a password

Username:

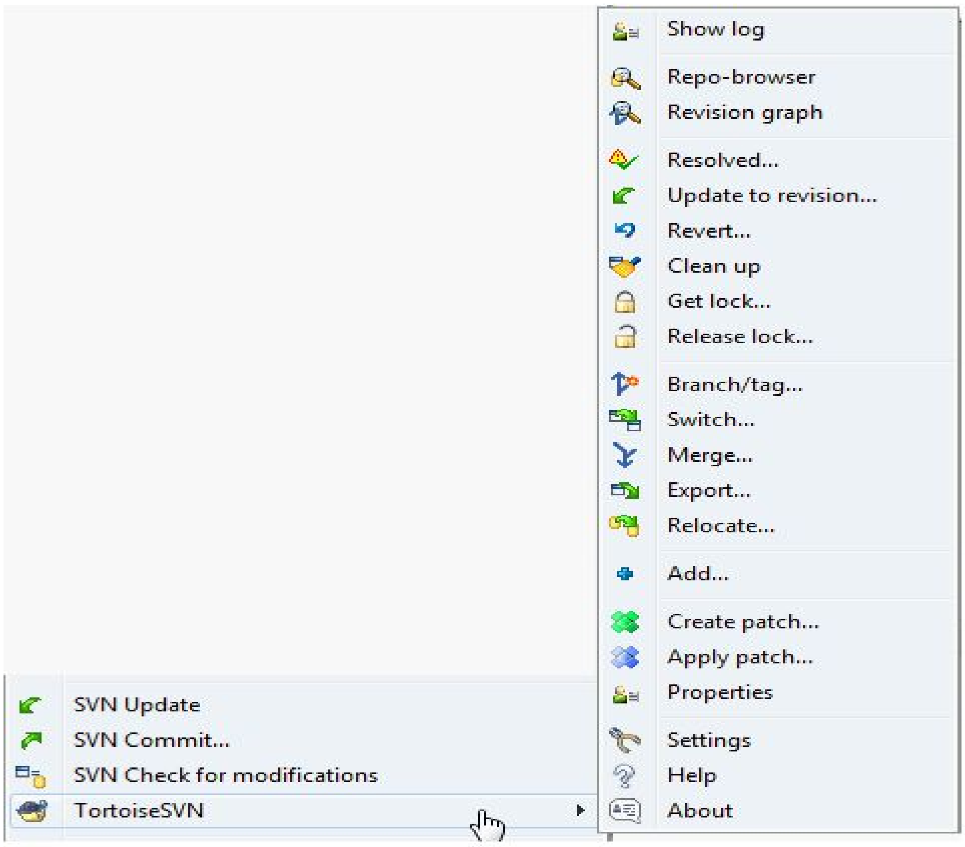
Password:

Save authentication

Note: when accessing Gameloft svn, please use username and password same as your windows account.

## Context Menus

All TortoiseSVN commands are invoked from the context menu of the windows explorer. Most are directly visible, when you right click on a file or folder. The commands that are available depend on whether the file or folder or its parent folder is under version control or not. You can also see the TortoiseSVN menu as part of the Explorer file menu.

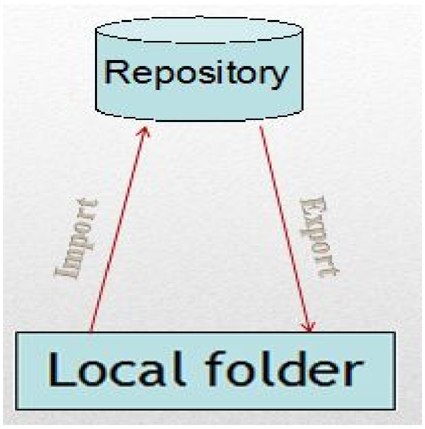


## Basic features

### Import & Export

**Import:** uploads a local file/folder/ project into the repository.

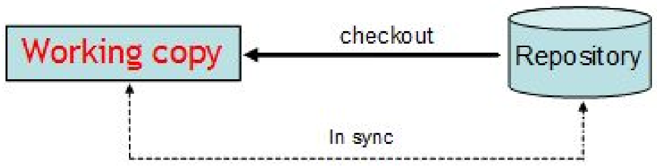
**Export:** opposite of import, give a local file/ folder/ project out the repository. It creates a clean directory tree without the version control metadata used in a working copy. Often used prior to publishing the contents.

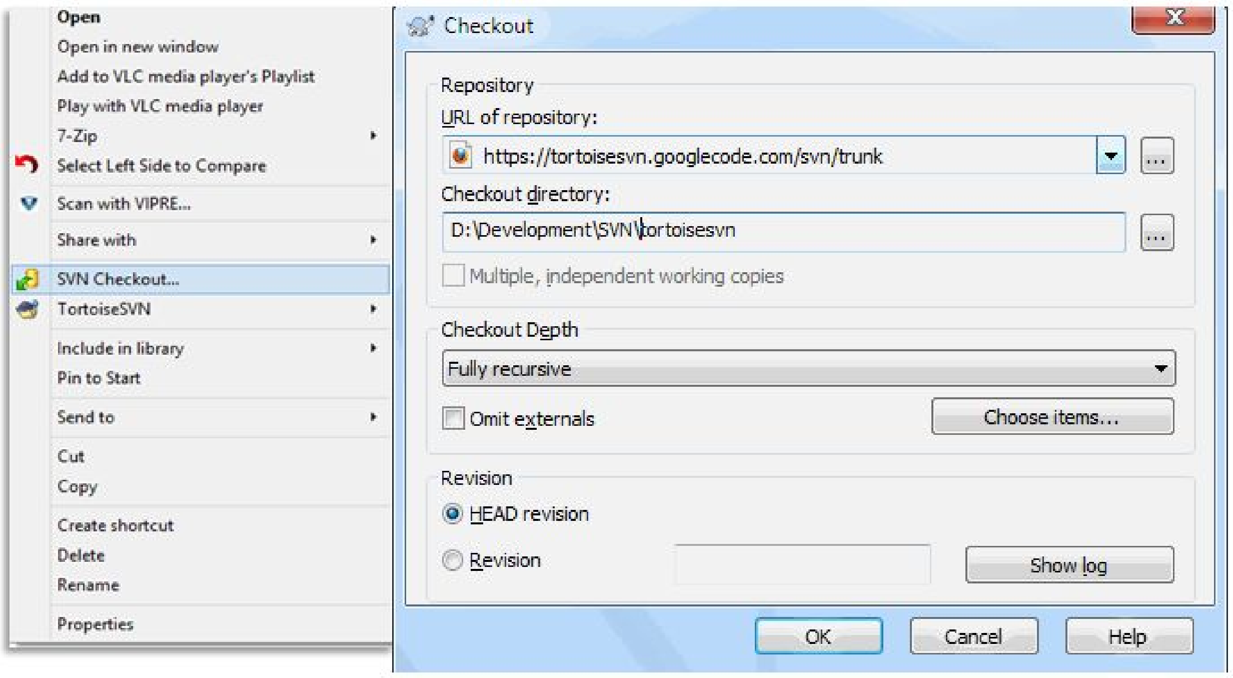


### Check-out

To creates a local working copy from the repository.

How: select a directory in windows explorer where you want to place your working copy. **Right click** to pop up the context menu and select the command **TortoiseSVN** → **Checkout...**, which brings up the following dialog box:





Repository
URL of repository: https://tortoisesvn.googlecode.com/svn/trunk
Checkout directory: D:\Development\SVN\tortoisesvn

Checkout Depth
Fully recursive

Revision
[✓ HEAD revision] Revision

Note: If you enter a folder name that does not yet exist, then a directory with that name is created.

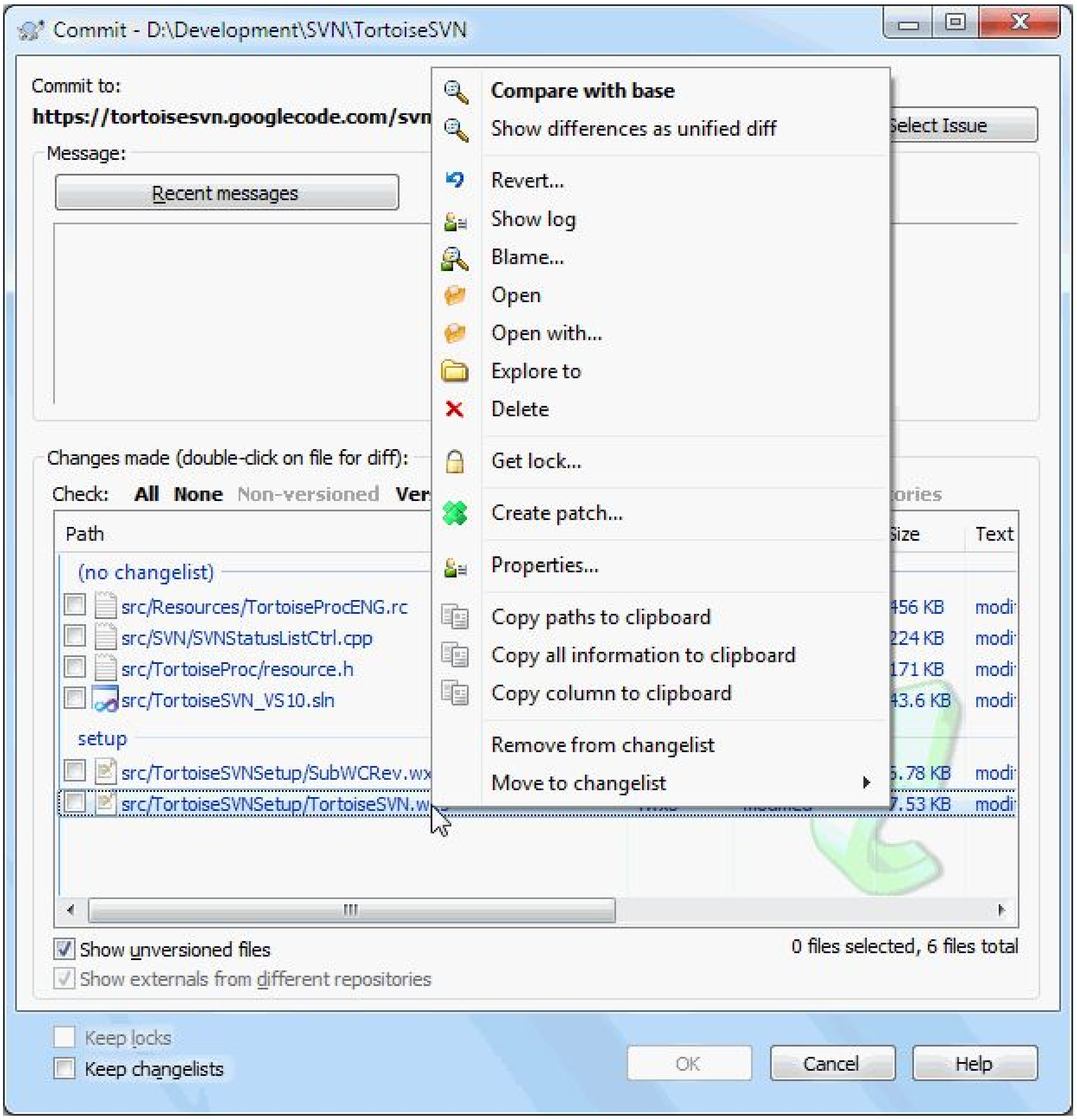
## Commit

### Commit:

Sending the changes you made to your working copy is known as *committing* the changes. But before you commit you have to make sure that your working copy is up to date. You can either use **TortoiseSVN** → **Update** directly. Or you can use **TortoiseSVN** → **Check for Modifications** first, to see which files have changed locally or on the server.

### Commit dialog:

If your working copy is up to date and there are no conflicts, you are ready to commit your changes. Select any file and/or folders you want to commit, then **TortoiseSVN** → **Commit....**

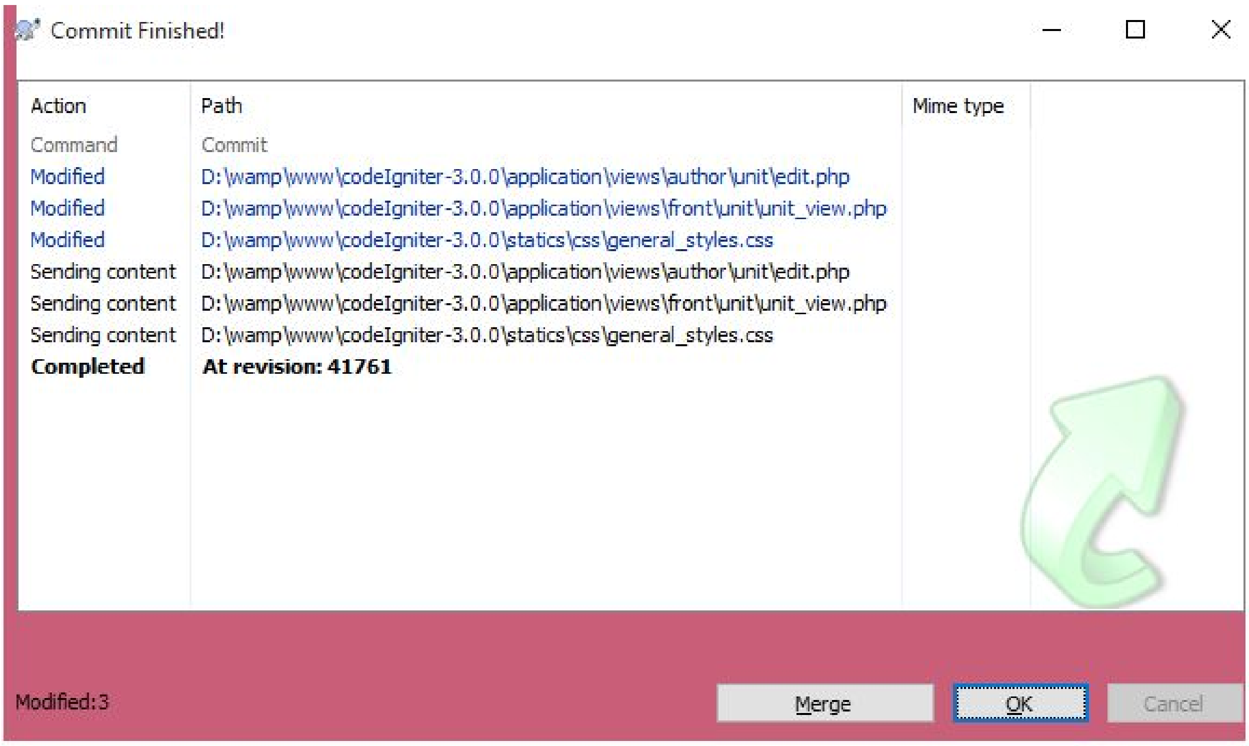

Commit - D:\Development\SVN\TortoiseSVN
Commit to:
https://tortoisesvn.googlecode.com/svn
Message:
Recent messages
Select Issue
Changes made (double-click on file for diff):
Check: All None Non-versioned Ver
Path
Size
Text
(no changelist)
src/Resources/TortoiseProcENG.rc
src/SVN/SVNStatusListCtrl.cpp
src/TortoiseProc/resource.h
src/TortoiseSVN_VS10.sln
setup
src/TortoiseSVNSetup/SubWCRev.wx
src/TortoiseSVNSetup/TortoiseSVN.w
Compare with base
Show differences as unified diff
Revert...
Show log
Blame...
Open
Open with...
Explore to
Delete
Get lock...
Create patch...
Properties...
Copy paths to clipboard
Copy all information to clipboard
Copy column to clipboard
Remove from changelist
Move to changelist
Show unversioned files
Show externals from different repositories
0 files selected, 6 files total
Keep locks
Keep changelists
OK
Cancel
Help

### Commit Log Messages

**Is a must enter a log message which describes the changes you are committing**. This will help you to see what happened and when, as you browse through the project log messages at a later date. The message can be as long or as brief as you like; Gameloft have guidelines for what should be (SVN rules).

### Commit progress:

After pressing OK, a dialog appears displaying the progress of the commit. Then you can see new revisions if commiting is sucessful

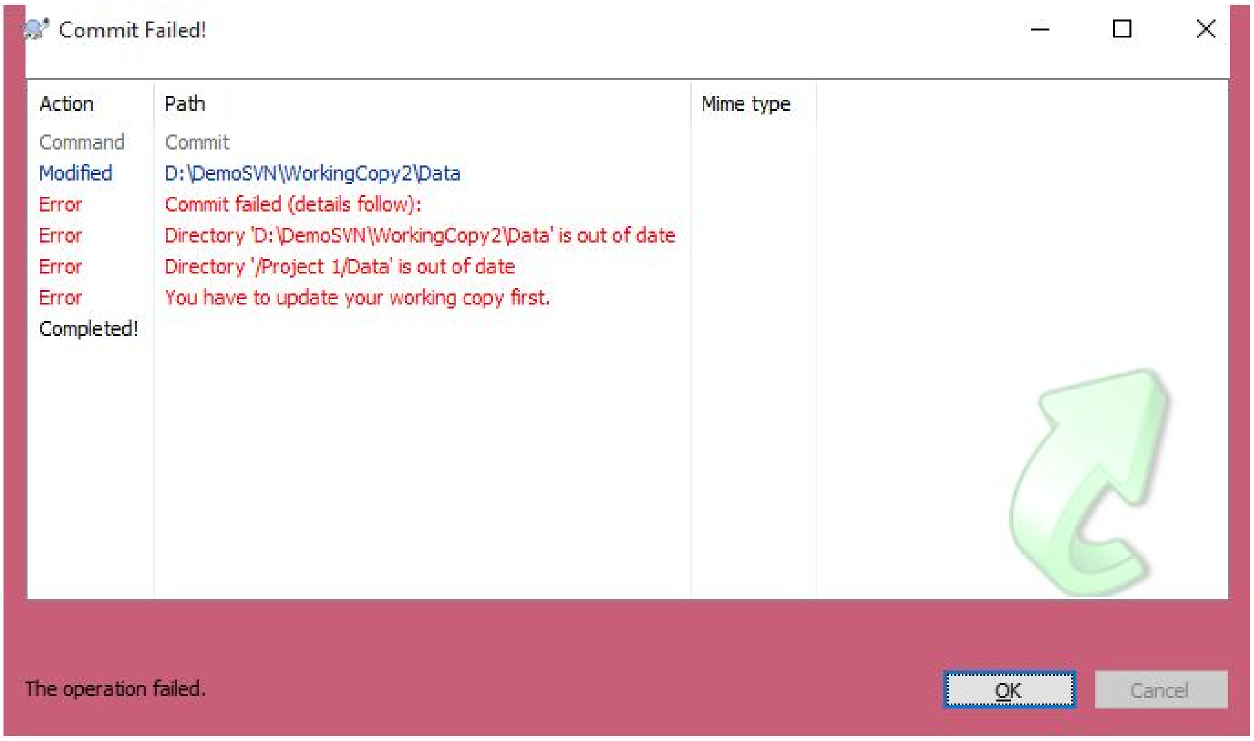


Commit Finished!

| Action | Path | Mime type |
|---|---|---|
| Command | Commit | |
| Modified | D:\wamp\www\codeIgniter-3.0.0\application\views\author\unit\edit.php | |
| Modified | D:\wamp\www\codeIgniter-3.0.0\application\views\front\unit\unit_view.php | |
| Modified | D:\wamp\www\codeIgniter-3.0.0\statics\css\general_styles.css | |
| Sending content | D:\wamp\www\codeIgniter-3.0.0\application\views\author\unit\edit.php | |
| Sending content | D:\wamp\www\codeIgniter-3.0.0\application\views\front\unit\unit_view.php | |
| Sending content | D:\wamp\www\codeIgniter-3.0.0\statics\css\general_styles.css | |
| **Completed** | **At revision: 41761** | |

Modified:3

And you will see the error messages if commit failed. Only carefully check the error messages you will find next actions to fix them.
Example:
**The Error message**: Directory"d:\DemoSVN\WorkingCopy2\Data' is out of date.
**Next action:** You have to update your working copy first.



Commit Failed!

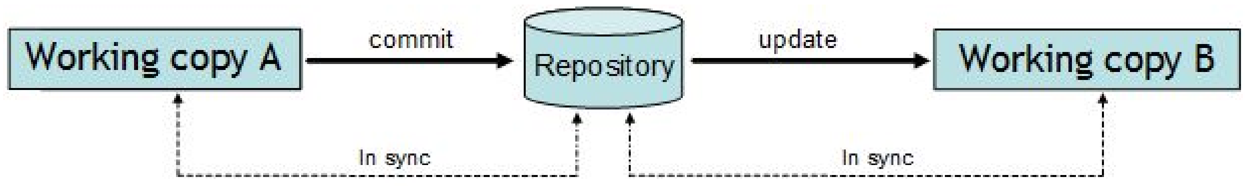
| Action | Path | Mime type |
|---|---|---|
| Command | Commit | |
| Modified | D:\DemoSVN\WorkingCopy2\Data | |
| Error | Commit failed (details follow): | |
| Error | Directory 'D:\DemoSVN\WorkingCopy2\Data' is out of date | |
| Error | Directory '/Project 1/Data' is out of date | |
| Error | You have to update your working copy first. | |
| Completed! | | |

The operation failed.

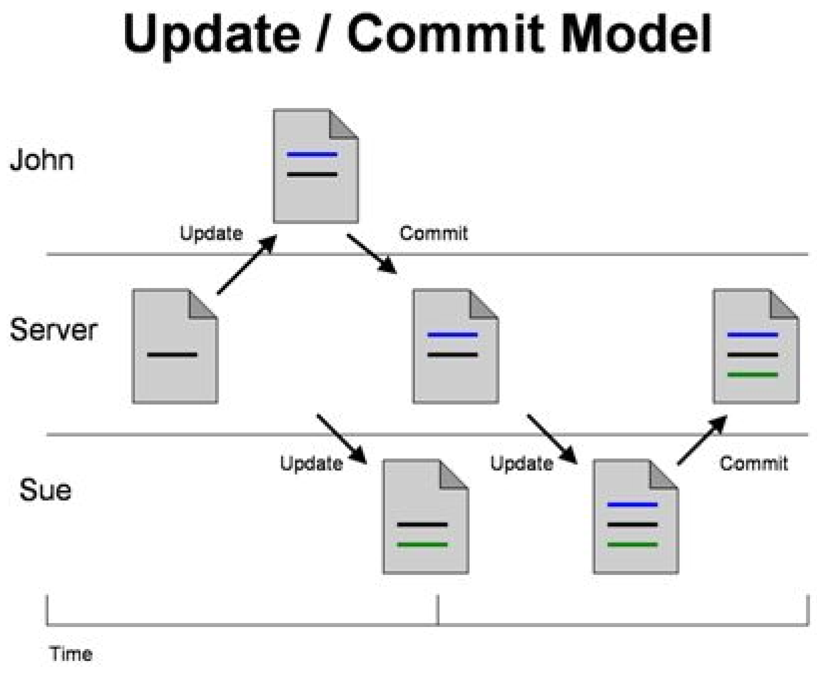
**Note: This is the most error message when committing without updating your working copy first.**

## Commit - Update

**Commit:** uploads changes made in the working copy to the repository. Every successful commit results in a new revision on the repository.



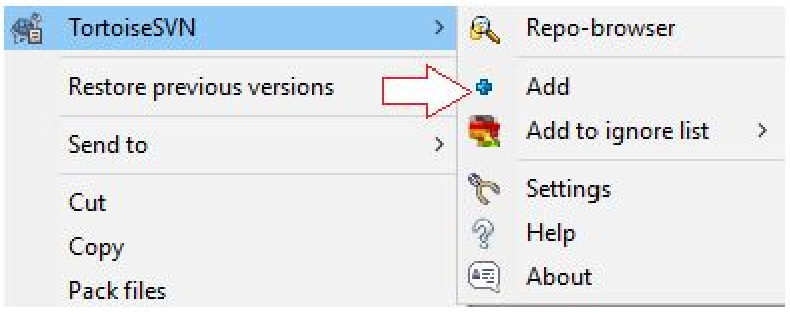
**Update:** brings the working copy in sync with the changes being made in the repository.



**Note: Always update your working copy before commiting.**

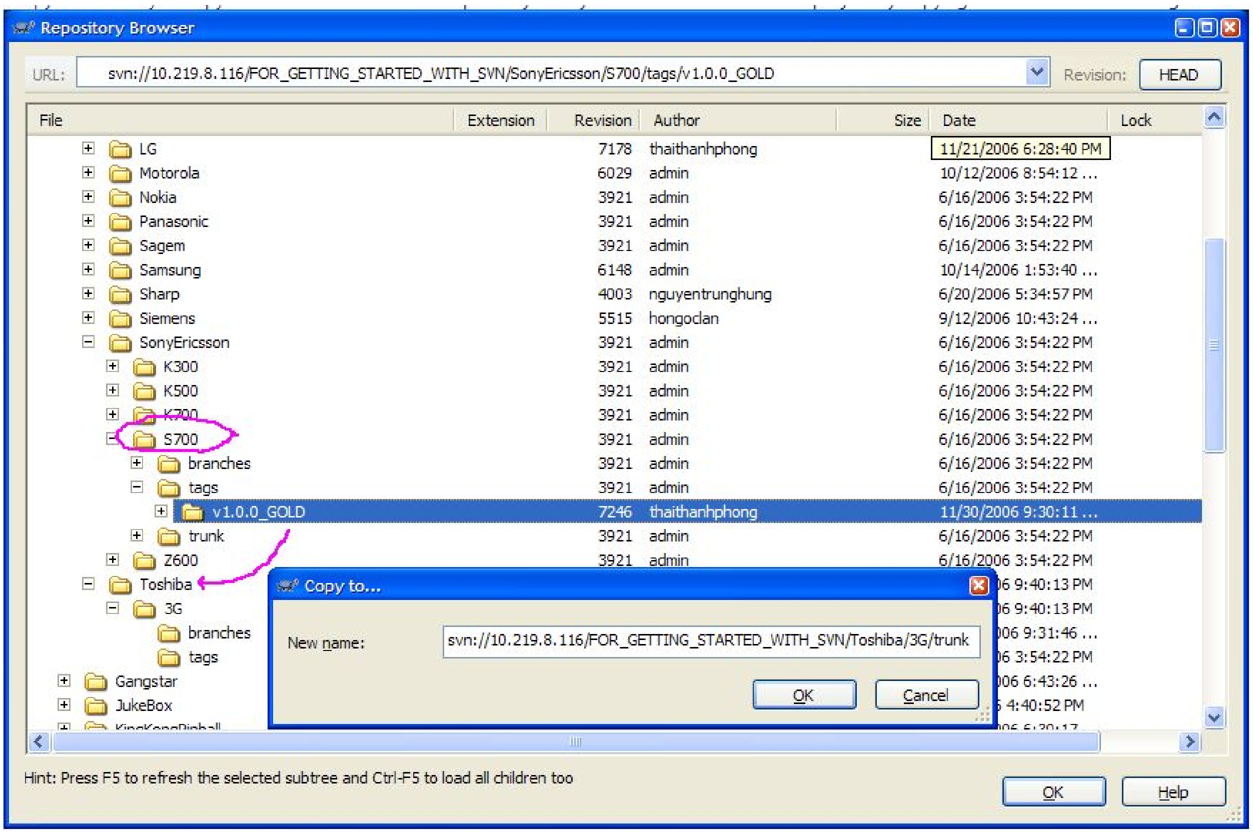
## Adding

If you created new files and/or directories during your development process then you need to add them to source control too. Select the file(s) and/or directory and use **TortoiseSVN → Add.** After you added the files/directories to source control the file appears with a **added icon overlay** which means you first have to commit your working copy to make those files/directories available to other developers. Adding a file/directory does *not* affect the repository!



## Copy to

"Copy to" means you copy it to a new location in the repository. And you need it to create a new project by copying from master source or gold source in porting team.


Repository Browser

URL: svn://10.219.8.116/FOR_GETTING_STARTED_WITH_SVN/SonyEricsson/S700/tags/v1.0.0_GOLD    Revision: HEAD

| File | Extension | Revision | Author | Size | Date | Lock |
|---|---|---|---|---|---|---|
| LG | | 7178 | thaithanhphong | | 11/21/2006 6:28:40 PM | |
| Motorola | | 6029 | admin | | 10/12/2006 8:54:12 ... | |
| Nokia | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| Panasonic | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| Sagem | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| Samsung | | 6148 | admin | | 10/14/2006 1:53:40 ... | |
| Sharp | | 4003 | nguyentrunghung | | 6/20/2006 5:34:57 PM | |
| Siemens | | 5515 | hongodan | | 9/12/2006 10:43:24 ... | |
| SonyEricsson | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| K300 | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| K500 | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| K700 | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| S700 | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| branches | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| tags | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| v1.0.0_GOLD | | 7246 | thaithanhphong | | 11/30/2006 9:30:11 ... | |
| trunk | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| Z600 | | 3921 | admin | | 6/16/2006 3:54:22 PM | |
| Toshiba | | | | | 06 9:40:13 PM | |
| 3G | | | | | 06 9:40:13 PM | |
| branches | | | | | 006 9:31:46 ... | |
| tags | | | | | 06 3:54:22 PM | |
| Gangstar | | | | | 006 6:43:26 ... | |
| JukeBox | | | | | 5 4:40:52 PM | |

Copy to...

New name: svn://10.219.8.116/FOR_GETTING_STARTED_WITH_SVN/Toshiba/3G/trunk

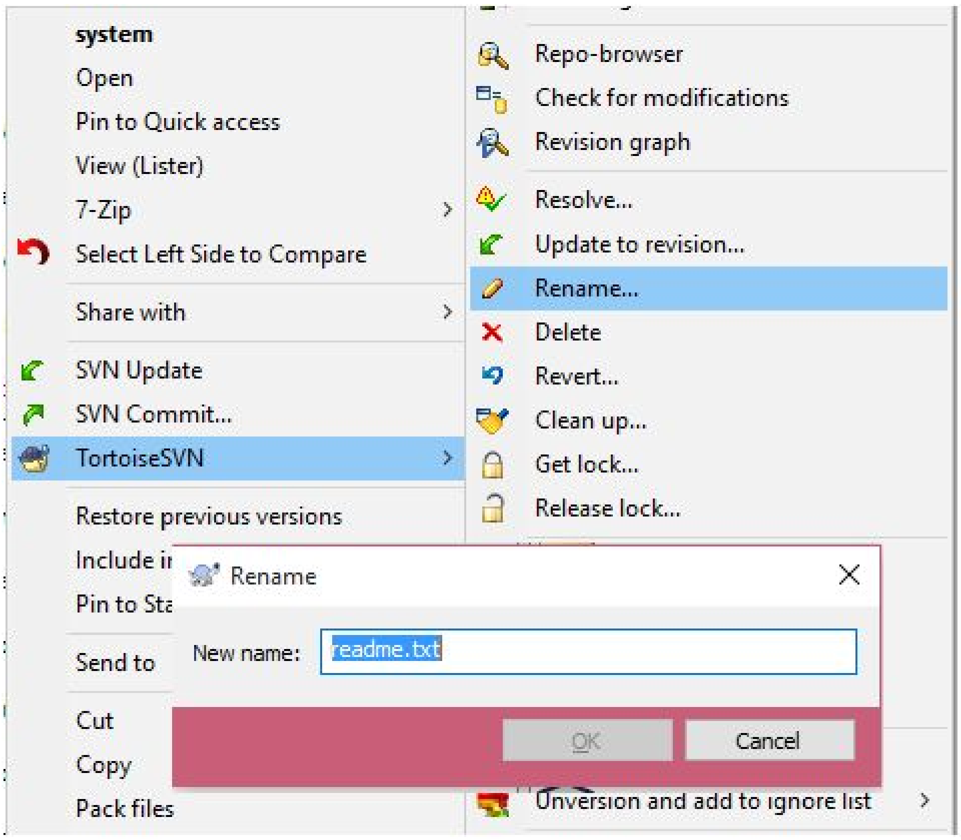
OK    Cancel

Hint: Press F5 to refresh the selected subtree and Ctrl-F5 to load all children too

OK    Help

## Renaming

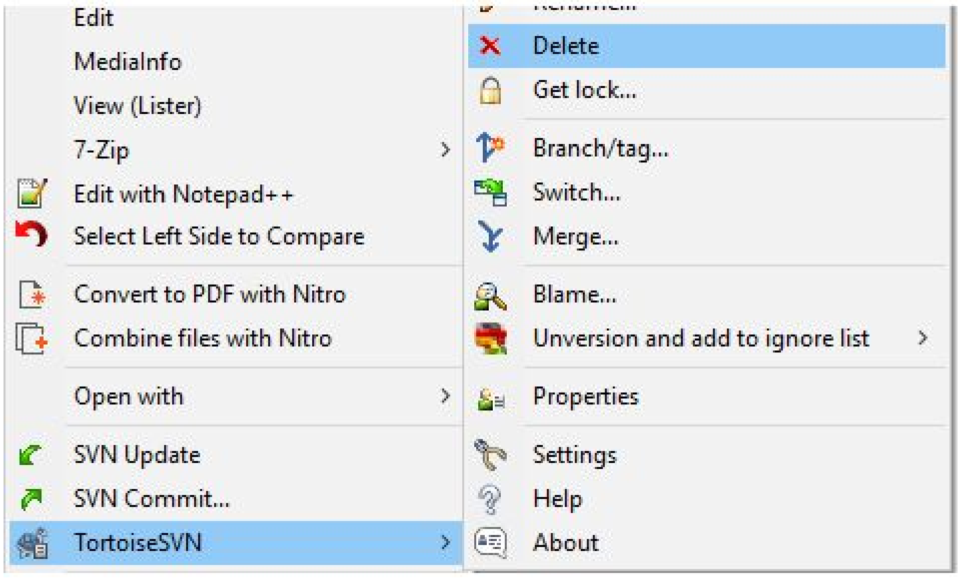
If you want to do a simple in-place rename of a file or folder, use **Context Menu** → **Rename**... Enter the new name for the item and you're done.



New name: readme.txt

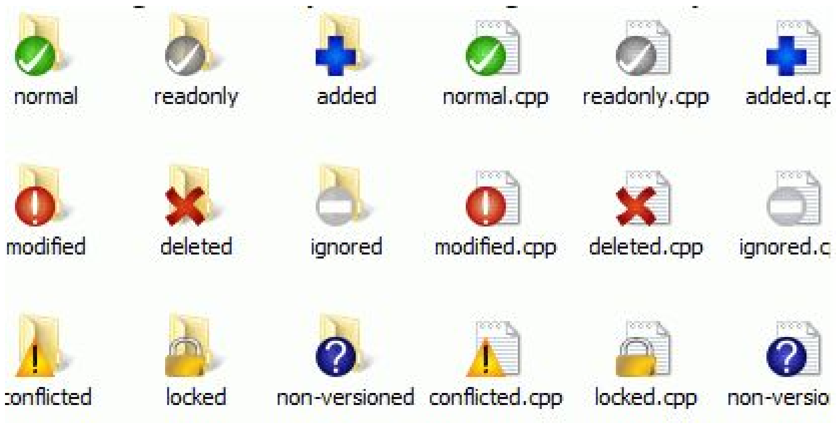
## Deleting

Use TortoiseSVN → Delete to remove files or folders from Subversion



## Getting status information - icons overlays

Show you at a glance which of your files have been modified.



✓ A fresh checked out working copy has a green checkmark as overlay. That means the Subversion status is *normal*.

! As soon as you start editing a file, the status changes to *modified* and the icon overlay then changes to a red exclamation mark. That way you can easily see which files were changed since you last updated your working copy and need to be committed.

⚠ If during an update a *conflict* occurs then the icon changes to a yellow exclamation mark.

● If you have set the *svn:needs-lock* property on a file, Subversion makes that file read-only until you get a lock on that file. Such files have this overlay to indicate that you have to get a lock first before you can edit that file.

🔒 If you hold a lock on a file, and the Subversion status is *normal*, this icon overlay reminds you that you should release the lock if you are not using it to allow others to commit their changes to the file.

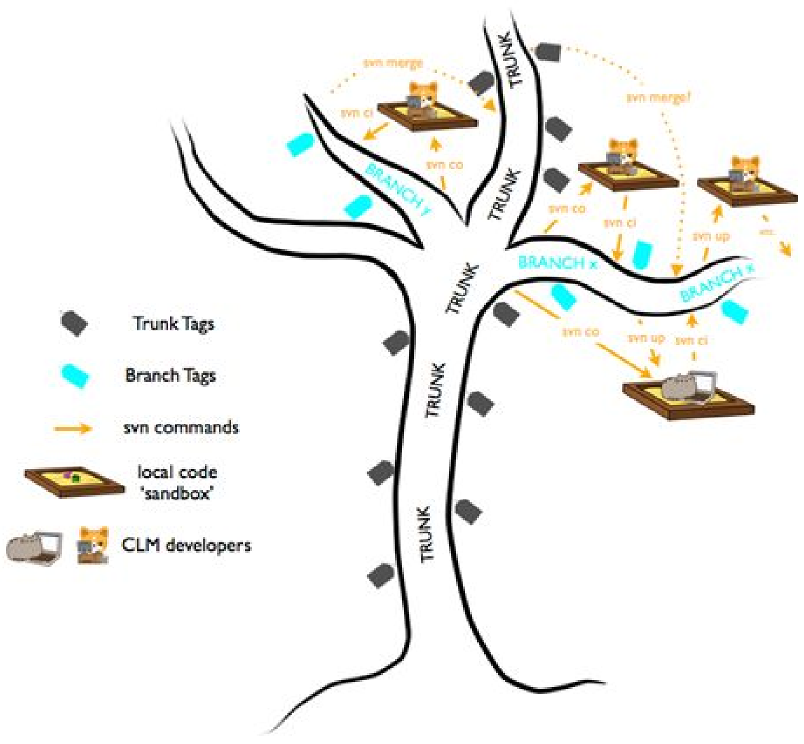
This icon shows you that some files or folders inside the current folder have been scheduled to be *deleted* from version control or a file under version control is missing in a folder.

The plus sign tells you that a file or folder has been scheduled to be *added* to version control.

The bar sign tells you that a file or folder is *ignored* for version control purposes. This overlay is optional.

This icon shows files and folders which are not under version control, but have not been ignored. This overlay is optional.

## Trunk, Branch and Tag



**Trunk:** would be the main body of development, originating from the start of the project until the present.
**Branch:**
We can isolate changes onto a separate line of development on SVN. This line is known as a ***branch***. Branches are often used to try out new features without disturbing the main line of development with compiler errors and bugs. As soon as the new feature is stable enough then the development branch is *merged* back into the main branch (trunk).
**Tag:**
Another feature of version control systems is the ability to mark particular revisions (e.g. a release version), so you can at any time recreate a certain build or environment. This process is known as ***tagging***.
Subversion does not have special commands for branching or tagging, but uses so-called "cheap copies" instead. Cheap copies are similar to hard links in Unix, which means that instead of making a complete copy in the repository, an internal link is created, pointing to a specific tree/revision. As a result branches and tags are very quick to create, and take up almost no extra space in the repository.

# SVN rules

## Prefix your log message using

| | |
|---|---|
| + | for a new feature |
| - | for a modified feature |
| * | for a bugfix |
| ! | for an important note |

## Write clear, useful log messages

**Bad:**

- Update  
* Fixed all bugs  
! Backup 23/10/2006

**Good:**

+ Added OTT sounds for Nokia MIDP1 + code to play them.  
- Updated IGP logo for WHAT'S NEW and BEST SELLERS.  
* Fixed bug #13754: wrong calculation of player's scores.

## Commit rules

+ Commit **soon**, commit **often**.  
+ Best to commit one conceptual change at a time: 1 bug fix or 1 feature implemented = 1 commit.  
+ Source code should at **least** compile.  
+ **Don't** put "garbages" (intermediate files, backup files, etc…) on the repo.

# Reference

## How to create a revision

A **svn commit** operation can publish changes to any number of files and directories as a single atomic transaction.
In your working copy, you can change files' contents, create, delete, rename and copy files and directories, and then commit the complete set of changes as a unit.
In the repository, each commit is treated as an atomic transaction: either all the commits changes take place, or none of them take place.
Each time the repository accepts a commit, this creates a new state of the filesystem tree, called a revision.
Each revision is assigned a unique natural number, one greater than the number of the previous revision.
The initial revision of a freshly created repository is numbered zero, and consists of nothing but an empty root directory.
A nice way to visualize the repository is as a series of trees. Imagine an array of revision numbers, starting at 0, stretching from left to right. Each revision number has a filesystem tree hanging below it, and each tree is a "snapshot" of the way the repository looked after each commit.

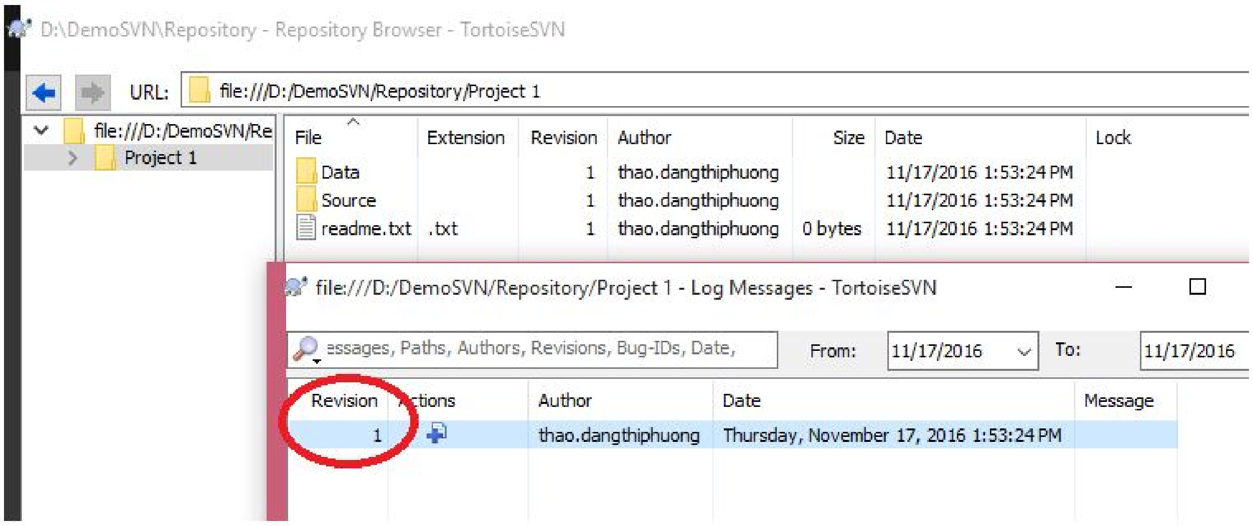
**Global Revision Numbers**

Unlike those of many other version control systems, Subversion of revision numbers apply to entire trees , not individual files. Each revision number selects an entire tree, a particular state of the repository after some committed change. Another way to think about it is that revision N represents the state of the repository filesystem after the Nth commit. When a Subversion user talks about ``revision 5 of foo.c", they really mean ``foo.c as it appears in revision 5." Notice that in general, revisions N and M of a file do not necessarily differ!

IMPORTANT NOTE: The working copies do not always correspond to any single revision in the repository; they may contain files from several different revisions.

**For example:**

**Step 1**: Suppose you create repository and import the first project on that repository. You will create revision 1 of the repository.



D:\DemoSVN\Repository - Repository Browser - TortoiseSVN

URL: file:///D:/DemoSVN/Repository/Project 1

| File | Extension | Revision | Author | Size | Date | Lock |
|---|---|---|---|---|---|---|
| Data | | 1 | thao.dangthiphuong | | 11/17/2016 1:53:24 PM | |
| Source | | 1 | thao.dangthiphuong | | 11/17/2016 1:53:24 PM | |
| readme.txt | .txt | 1 | thao.dangthiphuong | 0 bytes | 11/17/2016 1:53:24 PM | |

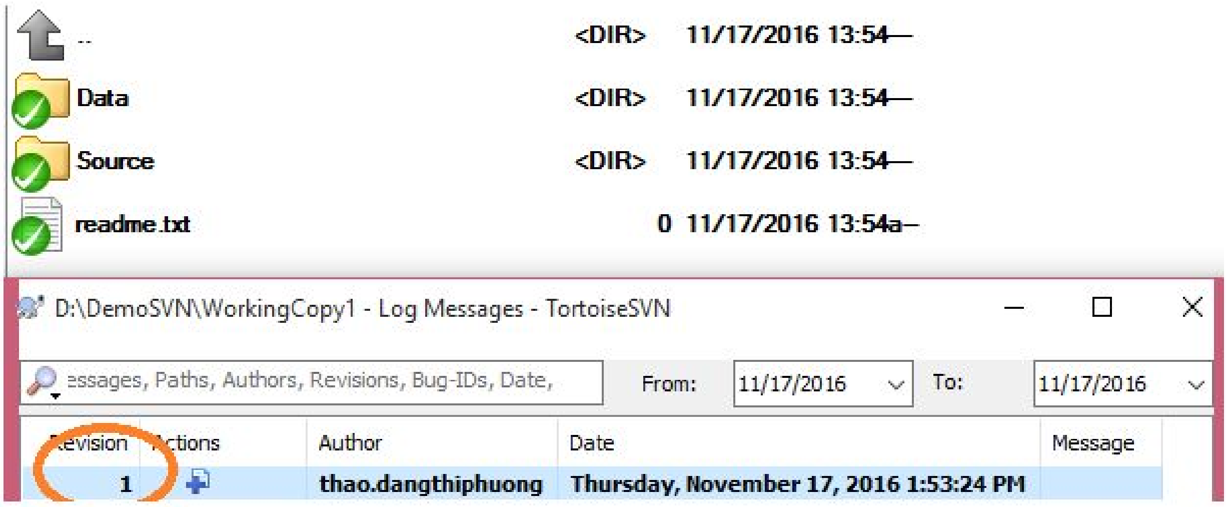
file:///D:/DemoSVN/Repository/Project 1 - Log Messages - TortoiseSVN

From: 11/17/2016 To: 11/17/2016

| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 1 | | thao.dangthiphuong | Thursday, November 17, 2016 1:53:24 PM | |

**Step 2:** You checkout a working copy to WorkingCopy1. And your working copy will now look like this:

**Step 2:** You checkout a working copy to WorkingCopy1. And your working copy will now look like this:



```
..            <DIR>  11/17/2016 13:54—
Data          <DIR>  11/17/2016 13:54—
Source        <DIR>  11/17/2016 13:54—
readme.txt        0  11/17/2016 13:54a–
```

D:\DemoSVN\WorkingCopy1 - Log Messages - TortoiseSVN

From: 11/17/2016 To: 11/17/2016

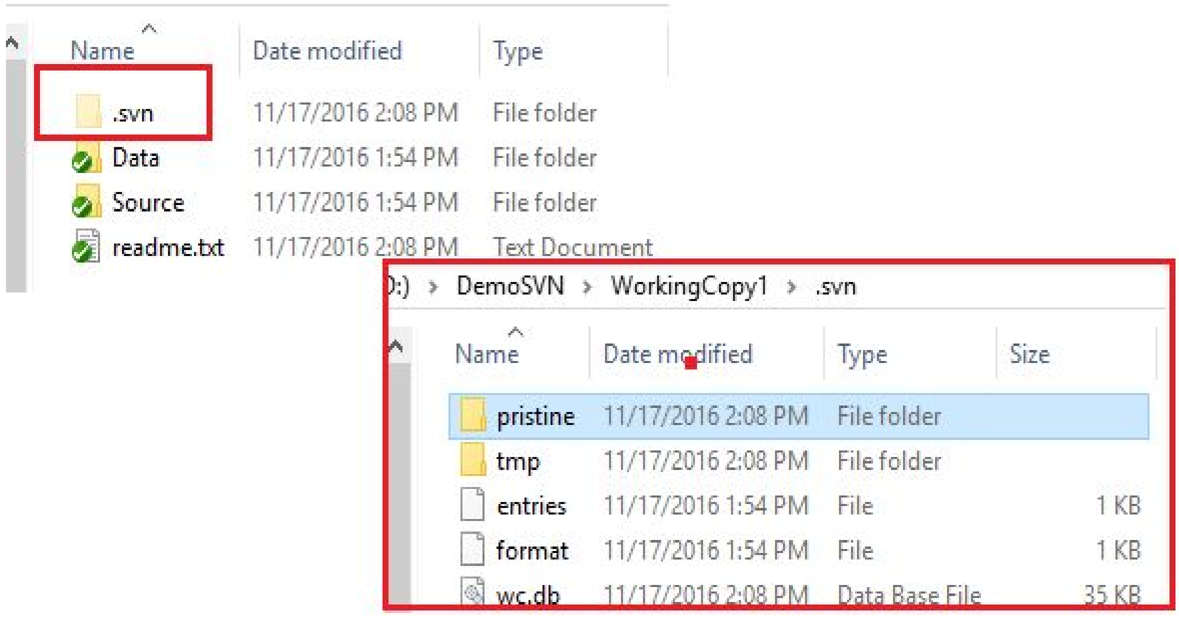
| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 1 | | thao.dangthiphuong | Thursday, November 17, 2016 1:53:24 PM | |

Data: 1
Source: 1
Readme.txt: 1

**Step 2**: you change content of readme file and commit that change. Assuming no other commits have taken place, your commit will create revision 2 of the repository, and your working copy will now look like this:
Data: 1
Source: 1
Readme.txt: 2
**Step 3**: Suppose you have a teammate, Sally works the same project. Now she checkouts the project, her revision of working copy is 2 ( newest revision). Then she adds game.cpp file in **source** folder commits, creating revision 3 of the repository.
Data: 1
Source: 3
Game.cpp: 3
Readme.txt: 2

## How Working Copies Track the Repositor:

For each file in a working directory, Subversion records two essential pieces of information in the .svn/ administrative area:
• what revision your working file is based on (this is called the file's working revision ), and
• a timestamp recording when the local copy was last updated by the repository.



| Name | Date modified | Type |
|---|---|---|
| .svn | 11/17/2016 2:08 PM | File folder |
| Data | 11/17/2016 1:54 PM | File folder |
| Source | 11/17/2016 1:54 PM | File folder |
| readme.txt | 11/17/2016 2:08 PM | Text Document |

):) › DemoSVN › WorkingCopy1 › .svn

| Name | Date modified | Type | Size |
|---|---|---|---|
| pristine | 11/17/2016 2:08 PM | File folder | |
| tmp | 11/17/2016 2:08 PM | File folder | |
| entries | 11/17/2016 1:54 PM | File | 1 KB |
| format | 11/17/2016 1:54 PM | File | 1 KB |
| wc.db | 11/17/2016 2:08 PM | Data Base File | 35 KB |

Given this information, by talking to the repository, Subversion can tell which of the following four states a working file is in:

**Unchanged, and current :**
The file is unchanged in the working directory, and no changes to that file have been committed to the repository since its working revision. A commit of the file will do nothing, and an update of the file will do nothing.

**Locally changed, and current**
The file has been changed in the working directory, and no changes to that file have been committed to the repository since its base revision. There are local changes that have not been committed to the repository, thus a commit of the file will succeed in publishing your changes, and an update of the file will do nothing.

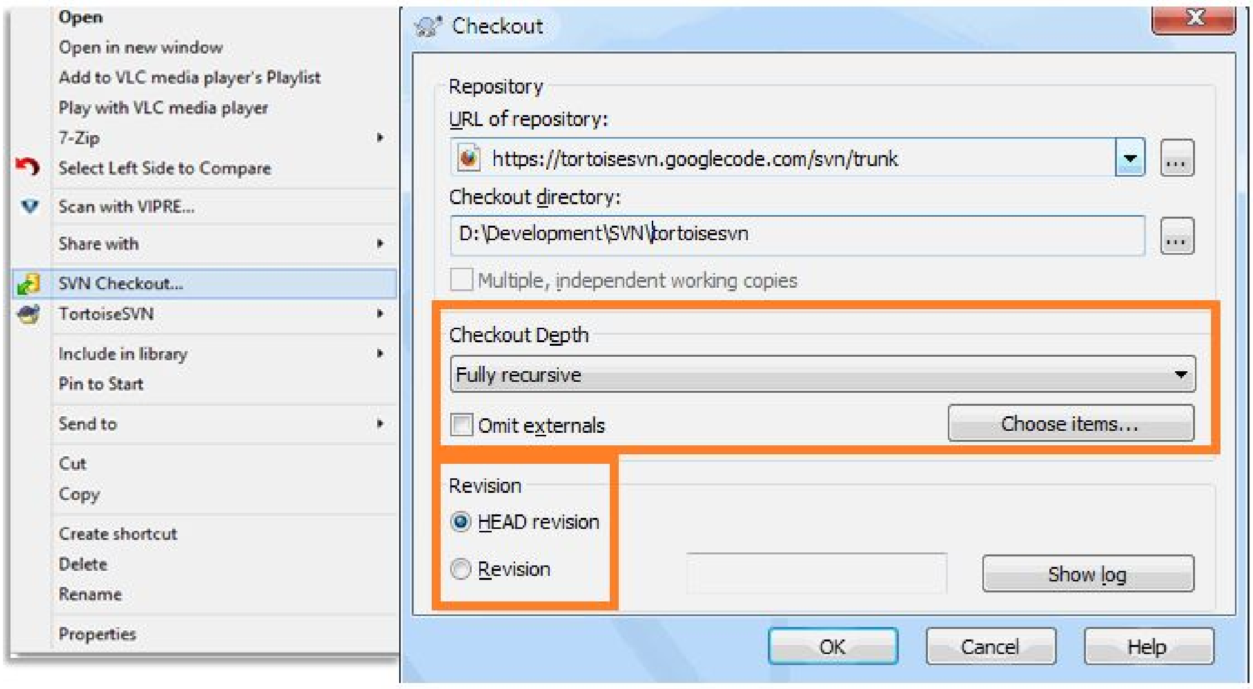
**Unchanged, and out-of-date**
The file has not been changed in the working directory, but it has been changed in the repository. The file should eventually be updated, to make it current with the public revision. A commit of the file will do nothing, and an update of the file will fold the latest changes into your working copy.

**Locally changed, and out-of-date**
The file has been changed both in the working directory, and in the repository. A commit of the file will fail with an out-of-date error. The file should be updated first; an update command will attempt to merge the public changes with the local changes. If Subversion can't complete the merge in a plausible way automatically, it leaves it to the user to resolve the conflict.

## Change settings for Checkout

Normally, you only need to checkout with default setting in Checkout dialog:
- Checkout Depth: Fully recursive.
- Revision: HEAD

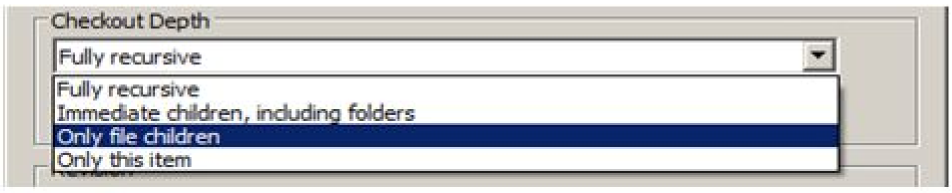
And click OK => You will have a working copy with new and full version.
For special case, you can easy set other settings for Checkout Depth or Revision.



URL of repository: https://tortoisesvn.googlecode.com/svn/trunk
Checkout directory: D:\Development\SVN\tortoisesvn
Checkout Depth: Fully recursive
Revision: [✓ HEAD revision] Revision

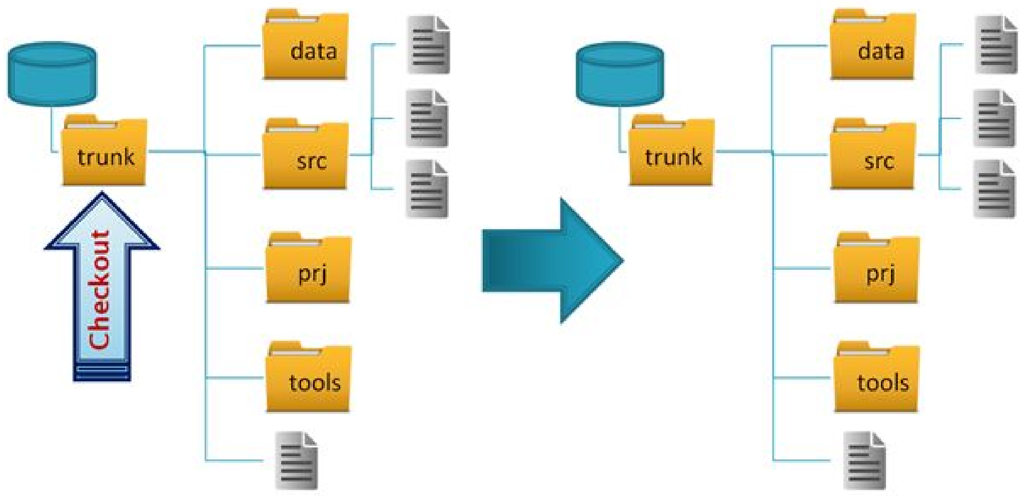
### Set Checkout Depth

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 checkout the top level folder only, then update selected folders recursively.


Checkout Depth
Fully recursive
Fully recursive
Immediate children, including folders
Only file children
Only this item

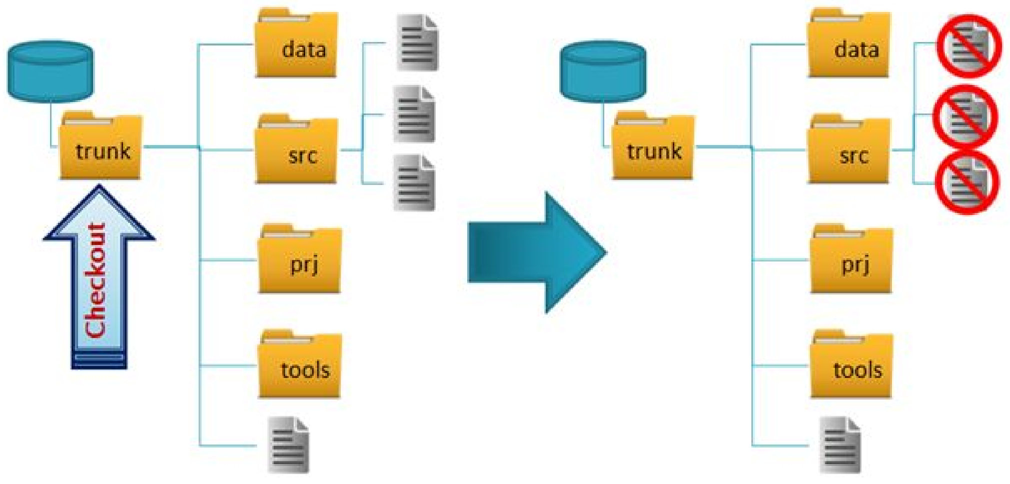
## Fully recursive

Checkout the entire tree, incluading all child folder and sub-folders.



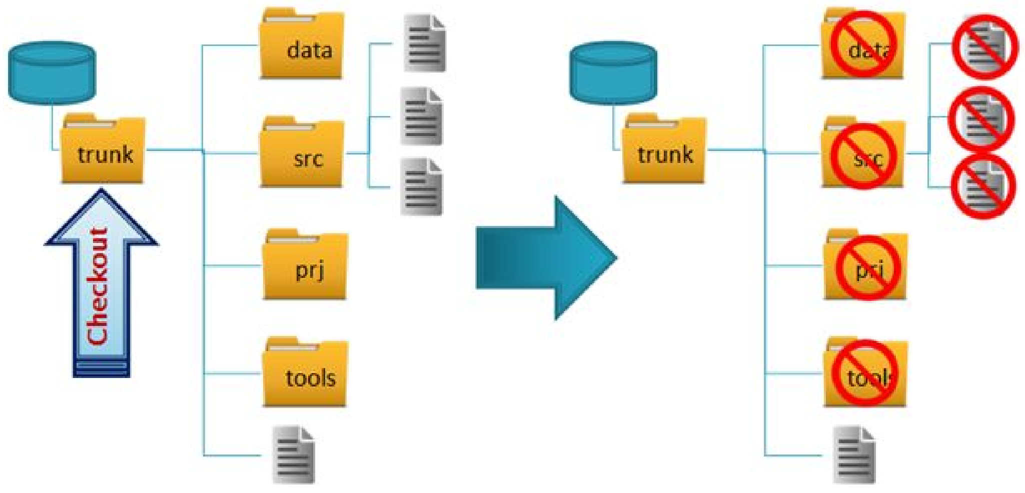
## Immediate children, including folders

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



## Only file children

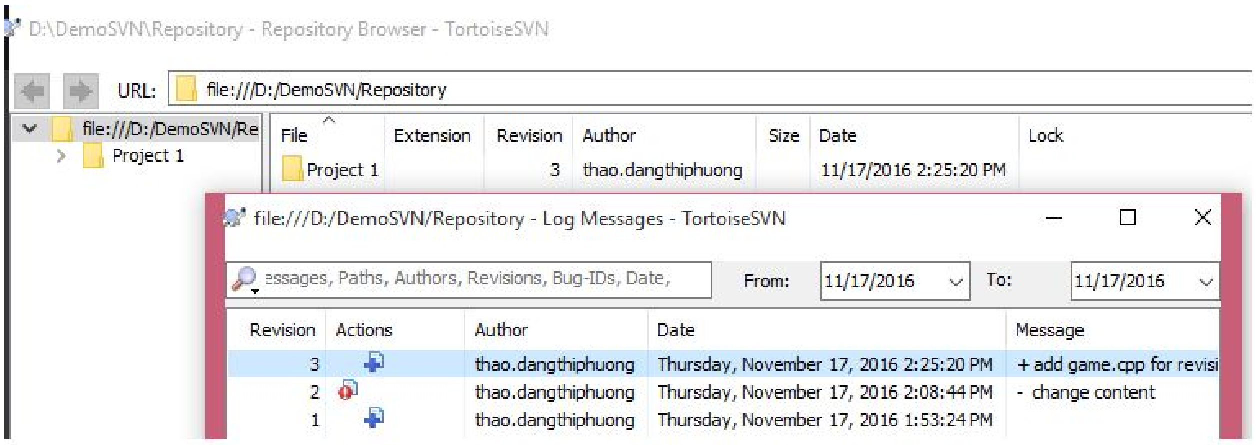
Checkout the specified directory, including all files but not do not checkout any child folders.



## Set revision:

You can choose the *revision* you want to checkout.

**Step 1**:  Show Log Message of project on repository. Find the revision to checkout.


D:\DemoSVN\Repository - Repository Browser - TortoiseSVN

URL: file:///D:/DemoSVN/Repository

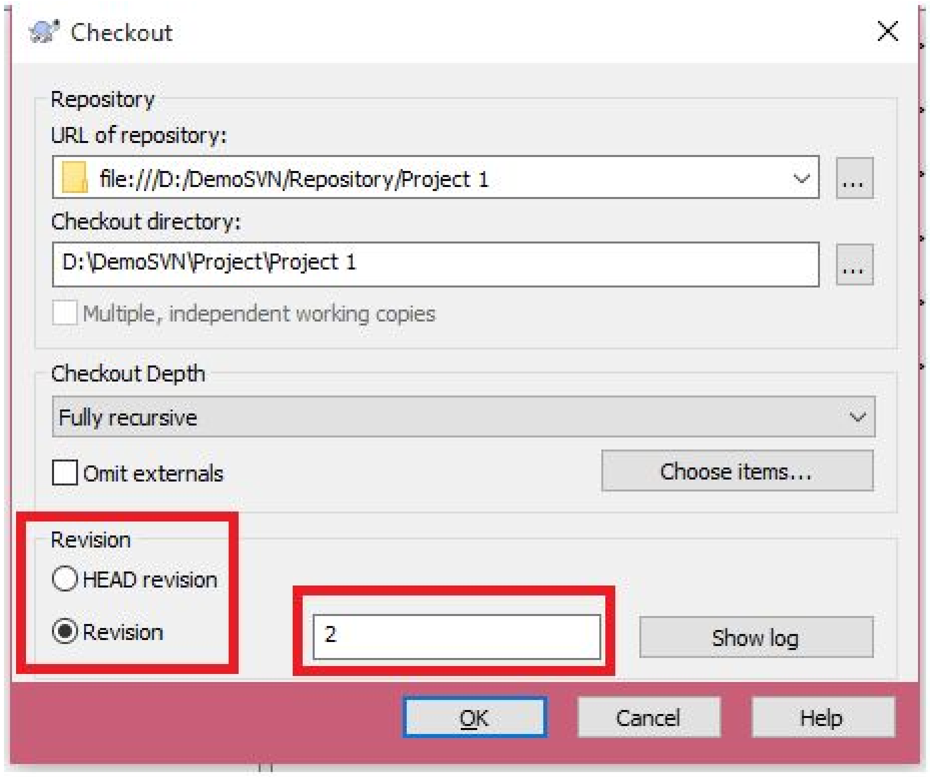
| File | Extension | Revision | Author | Size | Date | Lock |
|---|---|---|---|---|---|---|
| Project 1 | | 3 | thao.dangthiphuong | | 11/17/2016 2:25:20 PM | |

file:///D:/DemoSVN/Repository - Log Messages - TortoiseSVN

From: 11/17/2016 To: 11/17/2016

| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 3 | | thao.dangthiphuong | Thursday, November 17, 2016 2:25:20 PM | + add game.cpp for revisi |
| 2 | | thao.dangthiphuong | Thursday, November 17, 2016 2:08:44 PM | - change content |
| 1 | | thao.dangthiphuong | Thursday, November 17, 2016 1:53:24 PM | |

**Step 2:** Check **Revision** radio button and type number revision in the text box.And click OK button. You will have a working copy with a old revision.


Checkout

Repository

URL of repository: file:///D:/DemoSVN/Repository/Project 1

Checkout directory: D:\DemoSVN\Project\Project 1

Multiple, independent working copies

Checkout Depth: Fully recursive

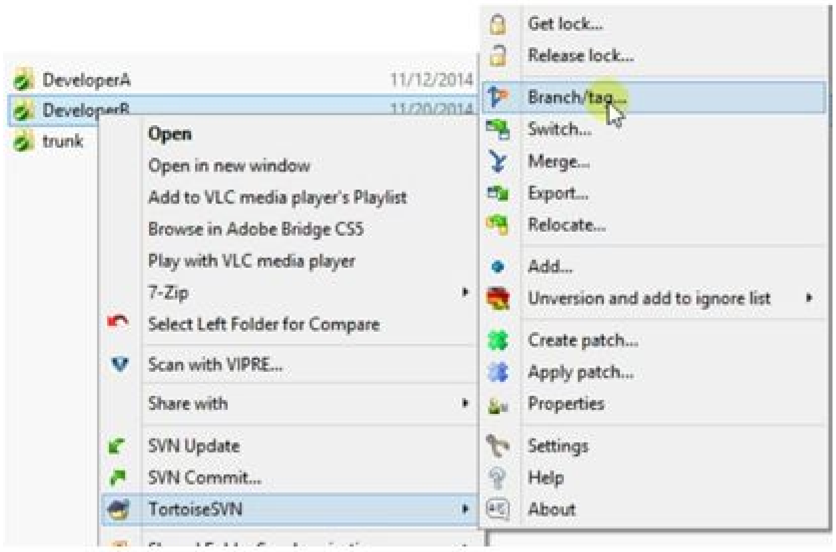
Omit externals

Revision: HEAD revision [✓ Revision] 2

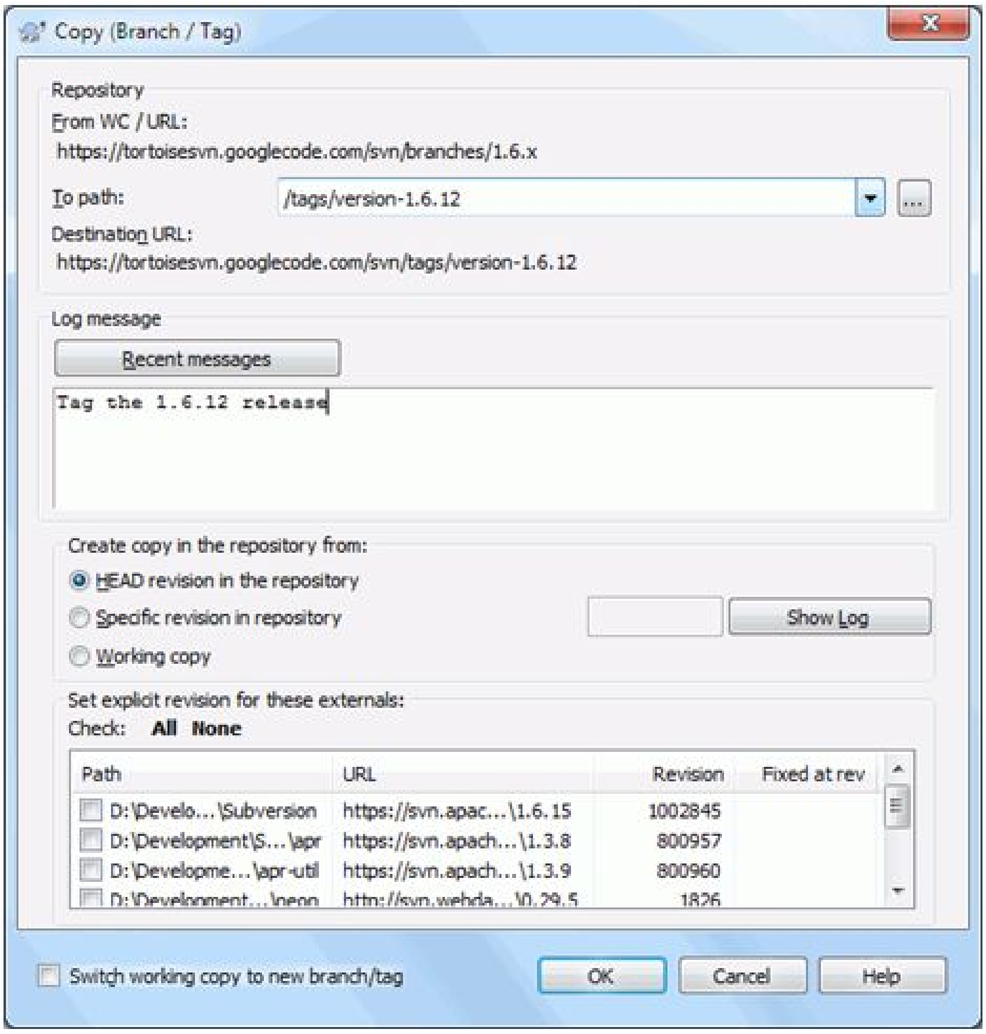
# Creating a branch or tag

## Having working copy - by Context Menu

The first step: is right click on desired versioned folder, then select TortoiseSVN →Branch/Tag....



The new window will need some information:



Copy (Branch / Tag)

Repository

From WC / URL:
https://tortoisesvn.googlecode.com/svn/branches/1.6.x

To path: /tags/version-1.6.12

Destination URL:
https://tortoisesvn.googlecode.com/svn/tags/version-1.6.12

Log message

Recent messages

```
Tag the 1.6.12 release
```

Create copy in the repository from:

(●) HEAD revision in the repository
( ) Specific revision in repository
( ) Working copy

Show Log

Set explicit revision for these externals:

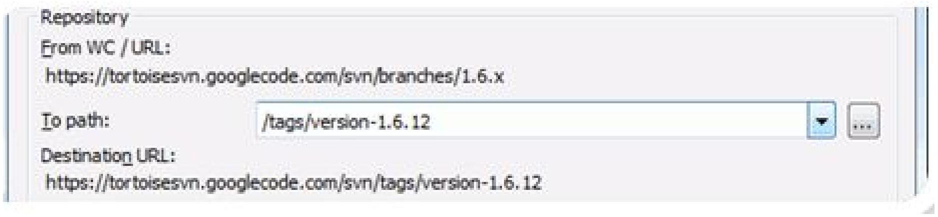
Check: **All** **None**

| Path | URL | Revision | Fixed at rev |
|---|---|---|---|
| ☐ D:\Develo...\Subversion | https://svn.apac...\1.6.15 | 1002845 | |
| ☐ D:\Development\S...\apr | https://svn.apach...\1.3.8 | 800957 | |
| ☐ D:\Developme...\apr-util | https://svn.apach...\1.3.9 | 800960 | |
| ☐ D:\Development...\neon | http://svn.webda...\0.29.5 | 1826 | |

☐ Switch working copy to new branch/tag

OK Cancel Help

## To path

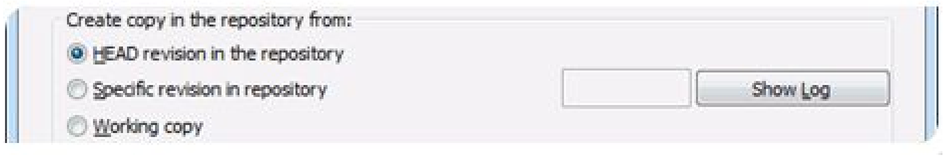

Repository
From WC / URL:
https://tortoisesvn.googlecode.com/svn/branches/1.6.x
To path: /tags/version-1.6.12
Destination URL:
https://tortoisesvn.googlecode.com/svn/tags/version-1.6.12

The default destination URL for the new branch will be the source URL on which your working copy is based. You will need to edit that URL to the new path for your branch/tag. So instead of:

  http://svn.collab.net/repos/ProjectName/trunk

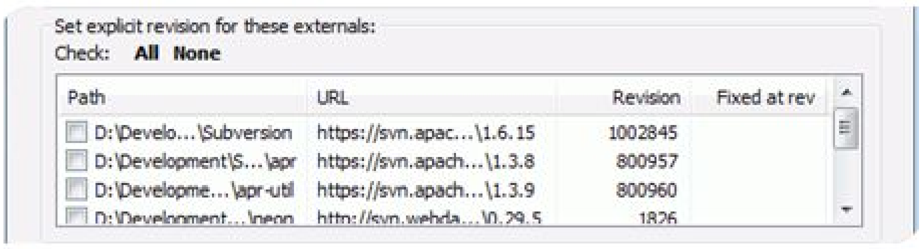
  You might now use something like:
  http://svn.collab.net/repos/ProjectName/tags/Release_1.10

## Select the source of the copy:


Create copy in the repository from:
[✓ HEAD revision in the repository]
Specific revision in repository
Working copy

HEAD revision in the repository The new branch is copied directly in the repository from the HEAD revision. No data needs to be transferred from your working copy, and the branch is created very quickly. Specific revision in the repository The new branch is copied directly in the repository but you can choose an older revision. This is useful if you forgot to make a tag when you released your project last week. If you can't remember the revision number, click the button on the right to show the revision log, and select the revision number from there. Again no data is transferred from your working copy, and the branch is created very quickly. Working copy The new branch is an identical copy of your local working copy. If you have updated some files to an older revision in your working copy, or if you have made local changes, that is exactly what goes into the copy. Naturally this sort of complex tag may involve transferring data from your working copy back to the repository if it does not exist there already.

## Set externals revisions:


Set explicit revision for these externals:
Check: **All** **None**

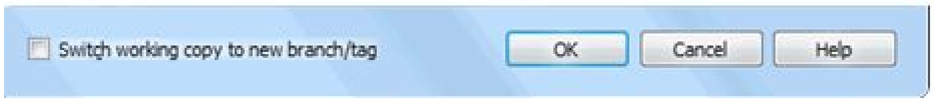
| Path | URL | Revision | Fixed at rev |
|---|---|---|---|
| ☐ D:\Develo...\Subversion | https://svn.apac...\1.6.15 | 1002845 | |
| ☐ D:\Development\S...\apr | https://svn.apach...\1.3.8 | 800957 | |
| ☐ D:\Developme...\apr-util | https://svn.apach...\1.3.9 | 800960 | |
| ☐ D:\Development...\neon | http://svn.webda...\0.29.5 | 1826 | |

If you want to make sure that the new tag always is in a consistent state, check all the externals to have their revisions pinned. If you don't check the externals and those externals point to a HEAD revision which might change in the future, checking out the new tag will check out that HEAD revision of the external and your tag might not compile anymore. So it's always a good idea to set the externals to an explicit revision when creating a tag. The externals are automatically pinned to either the current HEAD revision or the working copy BASE revision, depending on the source of the branch/tag:

| CopySource | Pinned Revision |
|---|---|
| HEAD revision in the repository | external repos HEAD revision |
| Specific revision in repository | external's repos HEAD revision |
| Working copy | external's working copy BASE revision |

**Note**: If a project that is included as an external has itself included externals, then those will not be tagged! Only externals that are direct children can be tagged.

### To checkout or switch:



If you want your working copy to be switched to the newly created branch automatically, use the **Switch working copy to new branch/tag** checkbox. But if you do that, first make sure that your working copy does not contain modifications. If it does, those changes will be merged into the branch working copy when you switch.

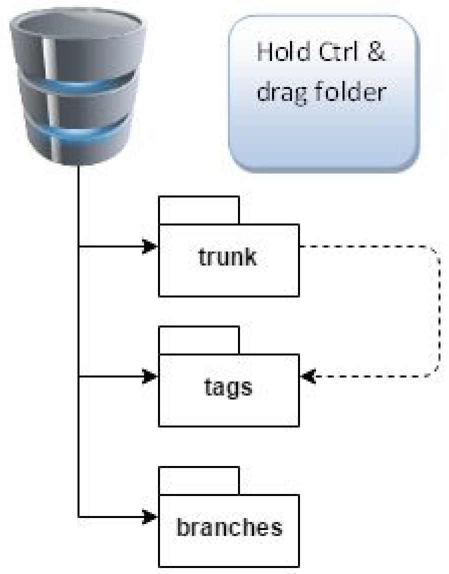
### Confirm changes pressing OK button.

**Note**: Unless you opted to switch your working copy to the newly created branch, creating a Branch or Tag does *not* affect your working copy. Even if you create the branch from your working copy, those changes are committed to the new branch, not to the trunk, so your working copy may still be marked as modified with respect to the trunk.

## Without having a working copy

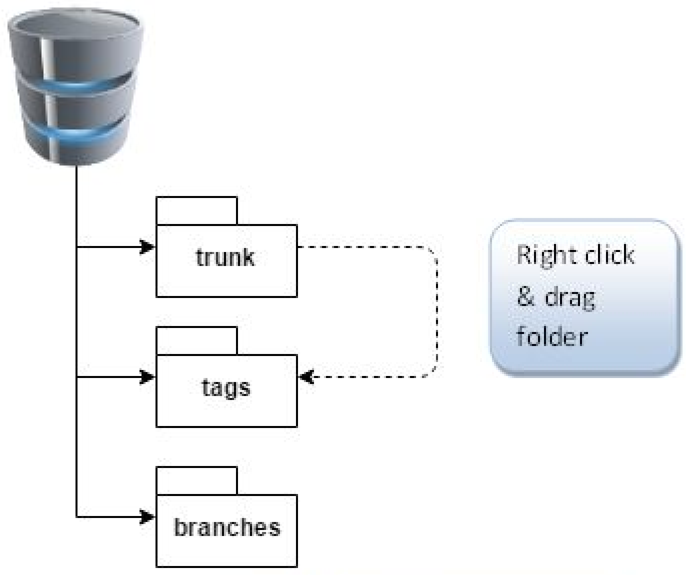
### Option A:

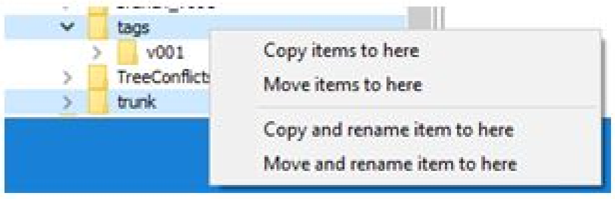
You can also create a branch or tag without having a working copy. To do that, open the repository browser. you can there drag folders to a new location. You have to hold down the CTRL key while you drag to create a copy, otherwise the folder gets moved, not copied.



### Option B:

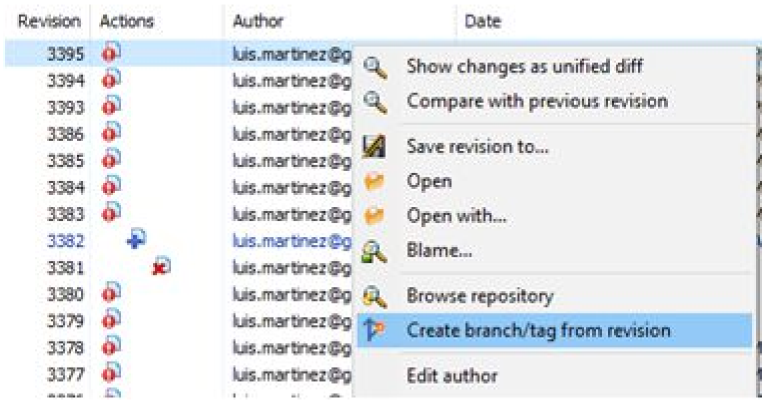
You can also drag a folder with the right mouse button. Once you release the mouse button you can choose from the context menu whether you want the folder to be moved or copied. Of course to create a branch or tag you must copy the folder, not move it.





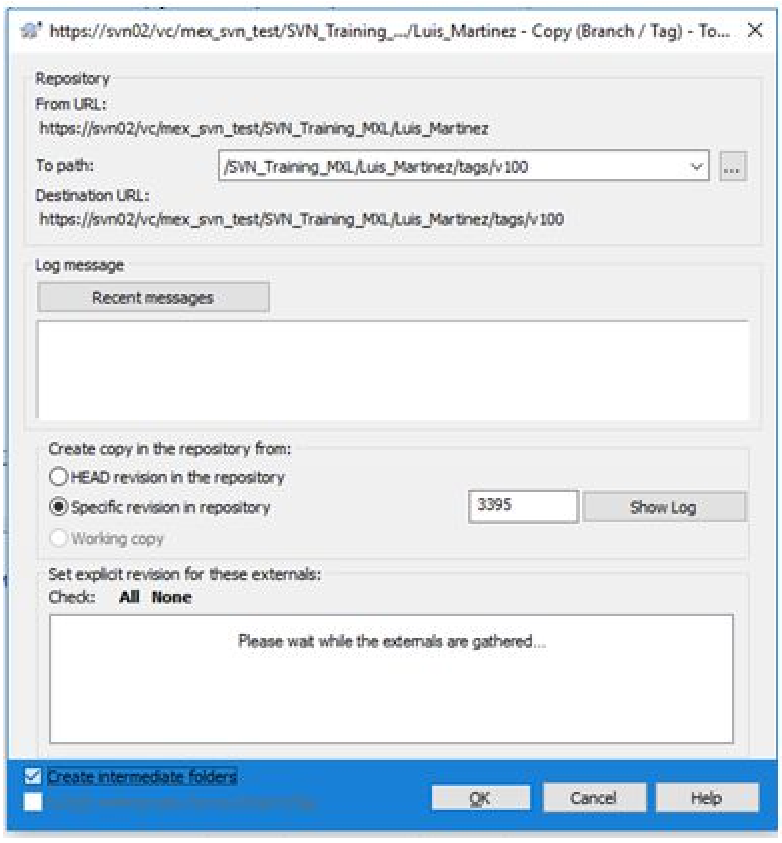
### Option C:

Yet another way is from the log dialog. You can show the log dialog for e.g. trunk, select a revision (either the HEAD revision at the very top or an earlier revision), right click and choose **create branch/tag from revision...**



| Revision | Author |
|---|---|
| 3395 | luis.martinez@g |
| 3394 | luis.martinez@g |
| 3393 | luis.martinez@g |
| 3386 | luis.martinez@g |
| 3385 | luis.martinez@g |
| 3384 | luis.martinez@g |
| 3383 | luis.martinez@g |
| 3382 | luis.martinez@g |
| 3381 | luis.martinez@g |
| 3380 | luis.martinez@g |
| 3379 | luis.martinez@g |
| 3378 | luis.martinez@g |
| 3377 | luis.martinez@g |

Then a new window will appear, add the branch/tag path, add the comment and set the other options, next click OK.



https://svn02/vc/mex_svn_test/SVN_Training_.../Luis_Martinez - Copy (Branch / Tag) - To...

Repository
From URL:
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez
To path: /SVN_Training_MXL/Luis_Martinez/tags/v100
Destination URL:
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/tags/v100

Log message
Recent messages

Create copy in the repository from:
HEAD revision in the repository
[✓ Specific revision in repository] 3395 Show Log
Working copy

Set explicit revision for these externals:
Check: **All** **None**
Please wait while the externals are gathered...

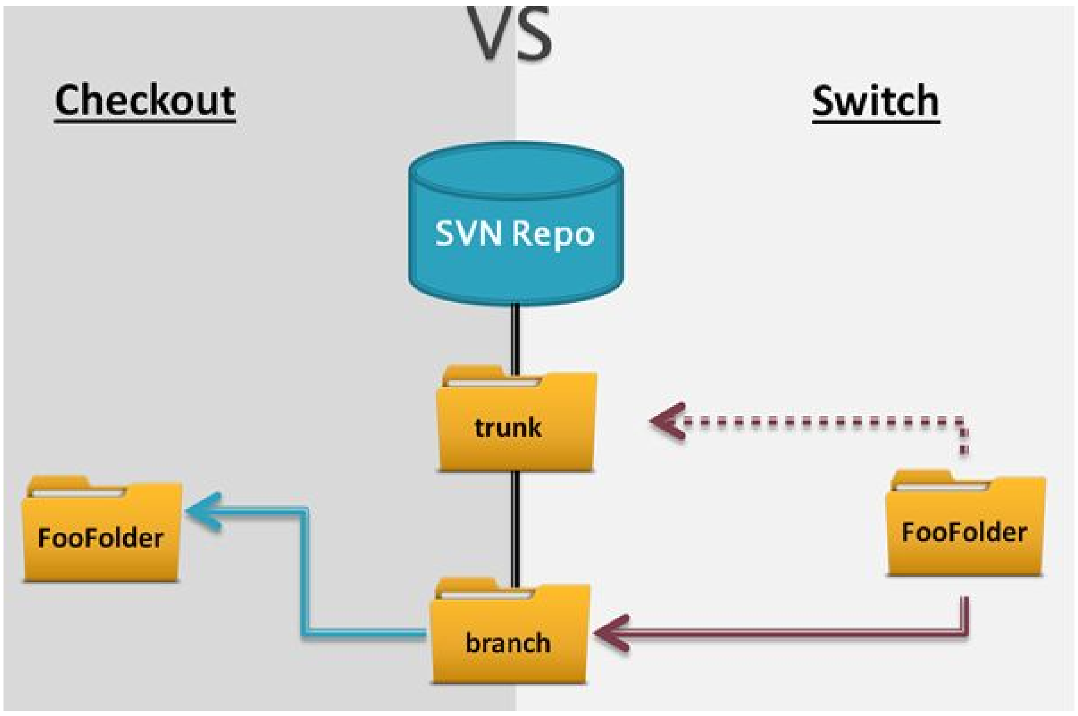
[✓ Create intermediate folders]

OK Cancel Help

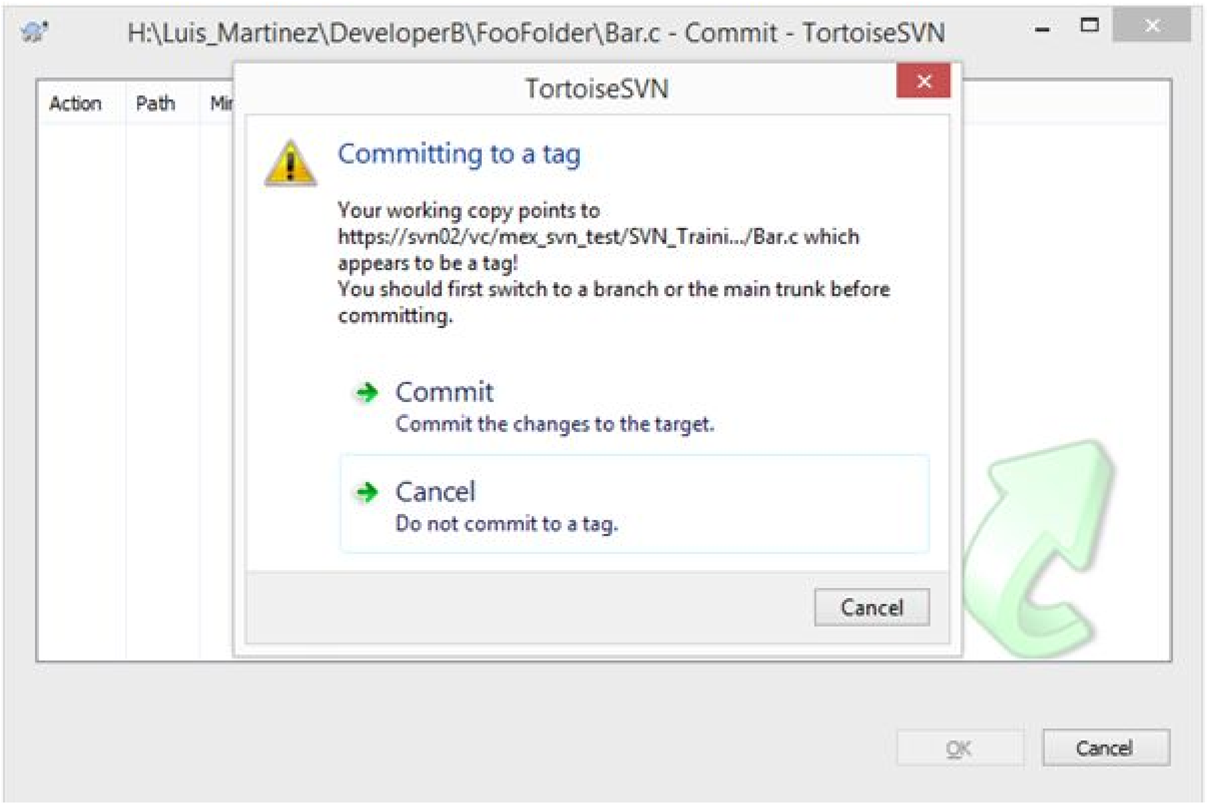
## Intermediate folders:

When you specify the target URL, all the folders up to the last one must already exist or you will get an error message. However if you want to create a branch/tag to an URL that has intermediate folders that don't exist yet you can check the option Create intermediate folders at the bottom of the dialog. If that option is activated, all intermediate folders are automatically created.

## To Checkout or to Switch



**Checkout**
Use **checkout** to make a fresh checkout in an empty folder. You can check out to any location on your local disk and you can create as many working copies from your repository as you like.
Switch
Switch your current working copy to the newly created copy in the repository. Again select the top level folder of your project and use **TortoiseSVN** → **Switch...** from the context menu.
Switch works just like Update in that it never discards your local changes. Any changes you have made to your working copy which have not yet been committed will be merged when you do the Switch. If you do not want this to happen then you must either commit the changes before switching, or revert your working copy to an already-committed revision (typically HEAD).
Commit on tag
Tags are typically used to create a static snapshot of the project at a particular stage. As such they are not normally used for development-that's what branches are for, which is reason we recommended the trunk/branches/tags repository structure in the first place. Working on tag revision is not a good idea, but because your local files are not write protected there is nothing to stop you doing this by mistake. However, if you try commit to a path in the repository which contains /tags/, TotoiseSVN will warn you.


H:\Luis_Martinez\DeveloperB\FooFolder\Bar.c - Commit - TortoiseSVN
Action
Path
TortoiseSVN
OK
Cancel

Committing to a tag

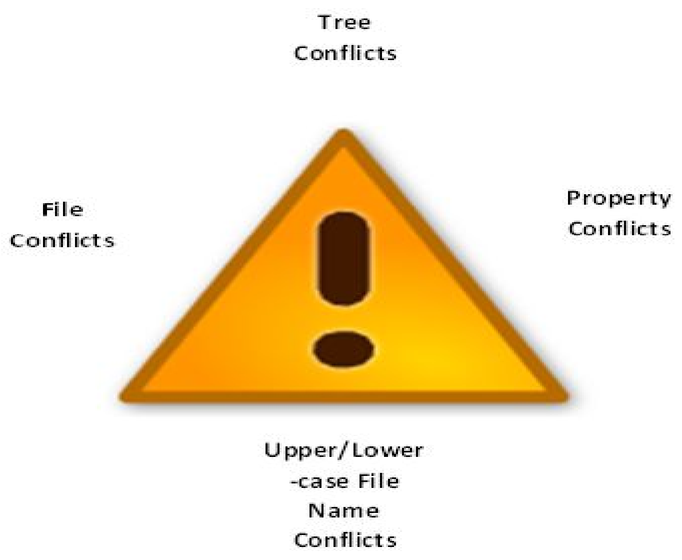
Your working copy points to https://svn02/vc/mex_svn_test/SVN_Traini.../Bar.c which appears to be a tag!
You should first switch to a branch or the main trunk before committing.

Commit
Commit the changes to the target.

Cancel
Do not commit to a tag.

How to:
It may be that you need to make further changes to a release which you have already tagged. The correct way to handle this is to create a new branch from the tag first and commit the branch. Do your Changes on this branch and then create a new tag from this new branch, e.g. Version_1.0.1.

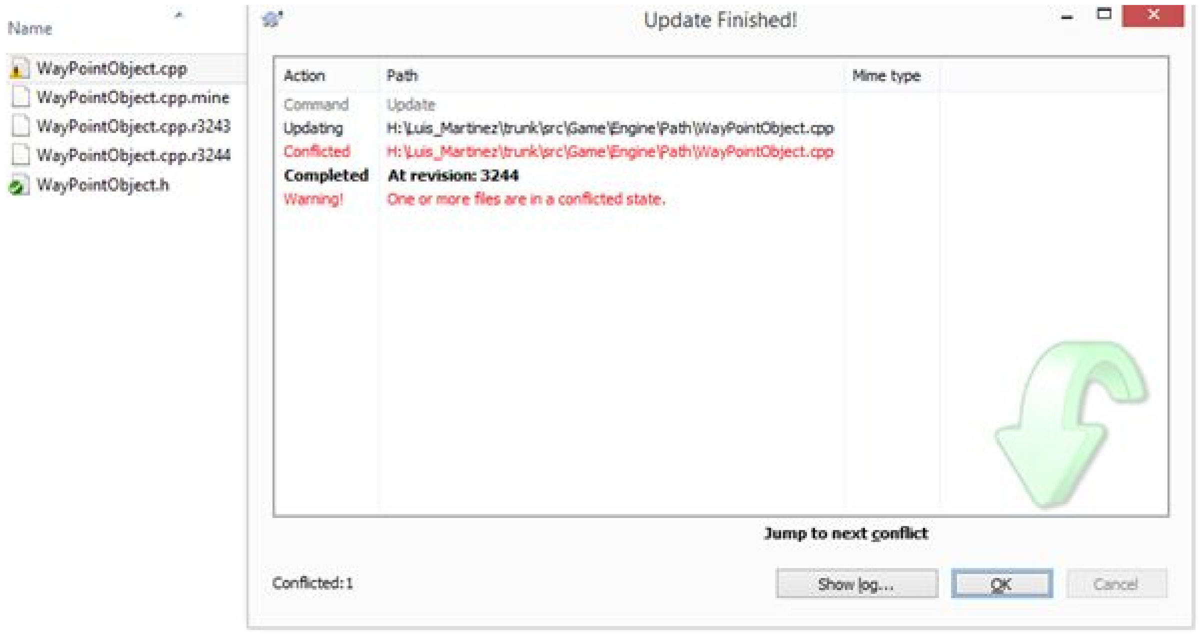
## Resolving conflicts



Once in a while, you will get a *conflict* when you update/merge your files from the repository or when you switch your working copy to a different URL. There are two kinds of conflicts:

### File conflicts:

A file conflict occurs if two (or more) developers have changed the same few lines of a file.



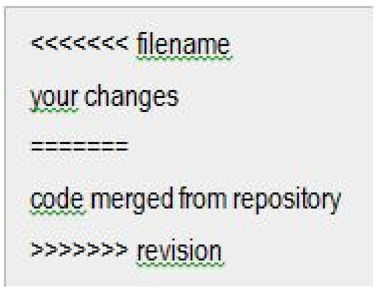
```
WayPointObject.cpp
WayPointObject.cpp.mine
WayPointObject.cpp.r3243
WayPointObject.cpp.r3244
WayPointObject.h
```

| Action | Path | Mime type |
|---|---|---|
| Command | Update | |
| Updating | H:\Luis_Martinez\trunk\src\Game\Engine\Path\WayPointObject.cpp | |
| Conflicted | H:\Luis_Martinez\trunk\src\Game\Engine\Path\WayPointObject.cpp | |
| Completed | At revision: 3244 | |
| Warning! | One or more files are in a conflicted state. | |

Conflicted: 1

WayPointObject.cpp

This is the conflicted file inside it's something like:

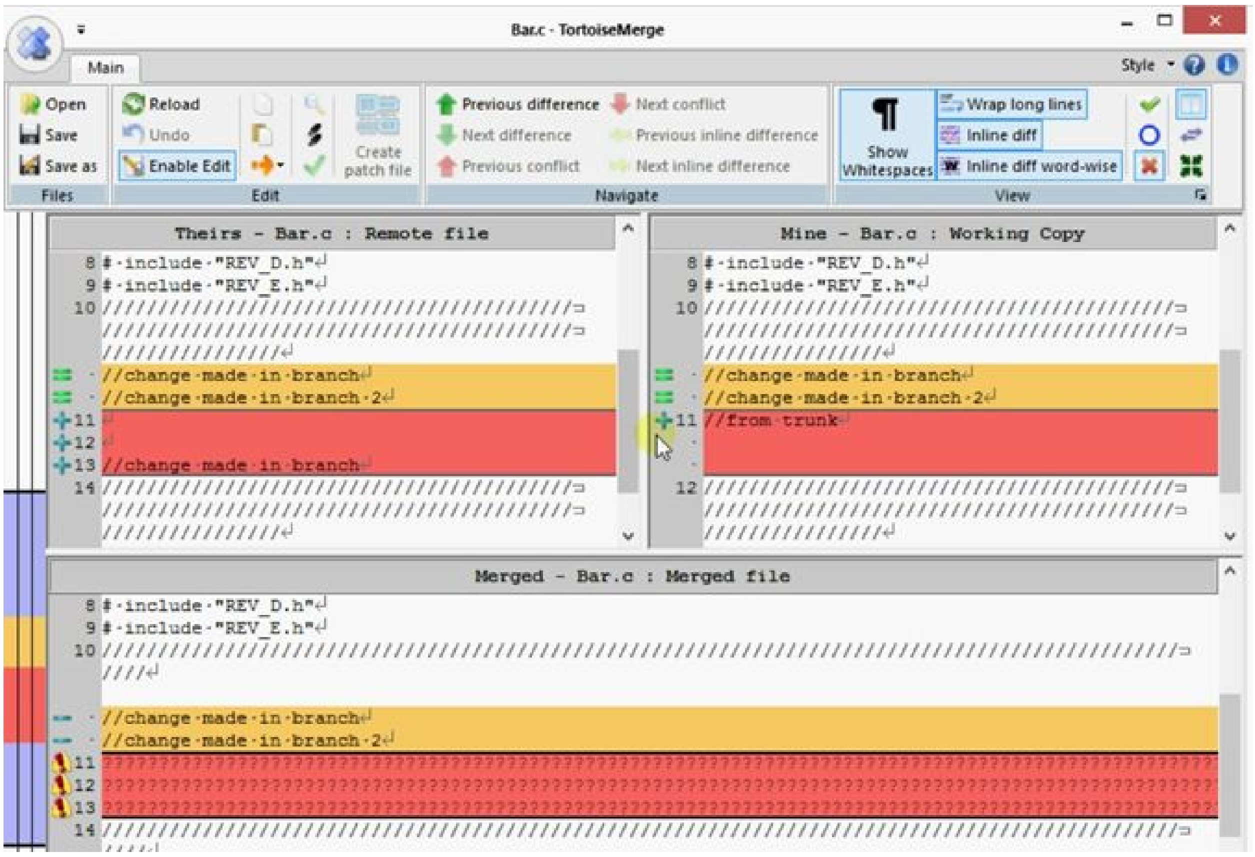


```
<<<<<<< filename
your changes
=======
code merged from repository
>>>>>>> revision
```

**WayPointObject.cpp.mine** is your file as it existed in your working copy before you updated your working copy - that is, without conflict markers. This file has your latest changes in it and nothing else.
**WayPointObject.cpp.r3243** This is the file that was the BASE revision before you updated your working copy. That is, it the file that you checked out before you made your latest edits.
**WayPointObject.cpp.r3244** This is the file that your Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

## How to resolve

You can either launch an external merge tool / conflict editor with **TortoiseSVN** → **Edit Conflicts** or you can use any text editor to resolve the conflict manually. You should decide what the code should look like, do the necessary changes and save the file.
- Using a merge tool is generally the easier option as they generally present the files involved in a 3-pane view and you don't have to worry about the conflict markers.

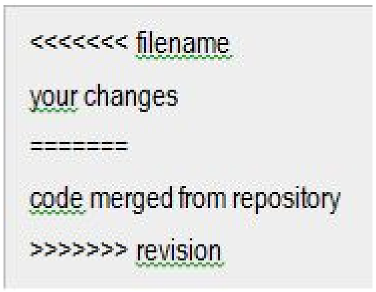


```
Theirs - Bar.c : Remote file
 8 #·include·"REV_D.h"
 9 #·include·"REV_E.h"
10 ////////////////////////////////////////////////
   ////////////////////////////////////////////////
   ////////////////
   //change·made·in·branch
   //change·made·in·branch·2
11
12
13 //change·made·in·branch
14 ////////////////////////////////////////////////
   ////////////////////////////////////////////////
   ////////////////
```

```
Mine - Bar.c : Working Copy
 8 #·include·"REV_D.h"
 9 #·include·"REV_E.h"
10 ////////////////////////////////////////////////
   ////////////////////////////////////////////////
   ////////////////
   //change·made·in·branch
   //change·made·in·branch·2
11 //from·trunk


12 ////////////////////////////////////////////////
   ////////////////////////////////////////////////
   ////////////////
```

```
Merged - Bar.c : Merged file
 8 #·include·"REV_D.h"
 9 #·include·"REV_E.h"
10 ////////////////////////////////////////////////////////////////////////////////////////////
   ////

   //change·made·in·branch
   //change·made·in·branch·2
11 ????????????????????????????????????????????????????????????????????????????????????????????
12 ????????????????????????????????????????????????????????????????????????????????????????????
13 ????????????????????????????????????????????????????????????????????????????????????????????
14 ////////////////////////////////////////////////////////////////////////////////////////////
```

- If you do use a text editor then you should search for lines starting with the string <<<<<<<.



```
<<<<<<< filename
your changes
=======
code merged from repository
>>>>>>> revision
```

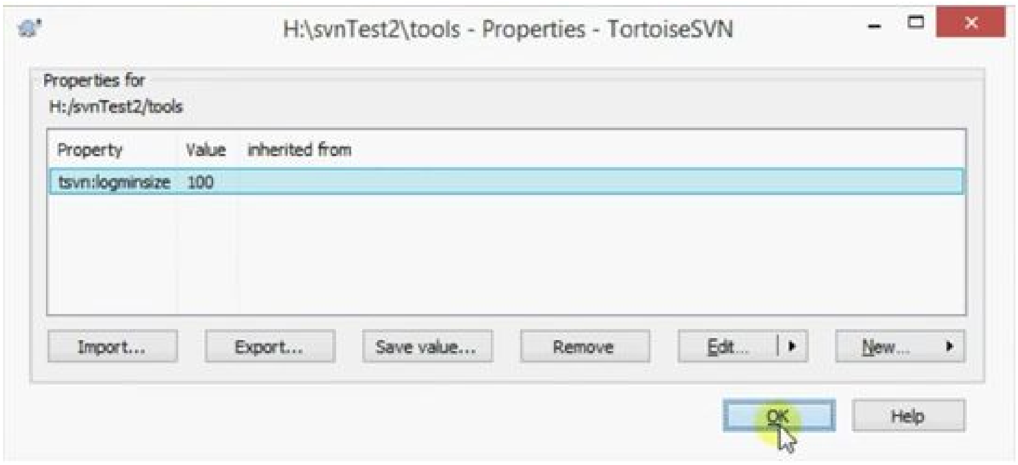
**Note**: Afterwards execute the command **TortoiseSVN** → **Resolved** and commit your modifications to the repository. Please note that the Resolve command does not really resolve the conflict. It just removes the filename.ext.mine and filename.ext.r* files, to allow you to commit your changes.

### Binary Files Conflict resolving

If you have conflicts with binary files, Subversion does not attempt to merge the files itself. The local file remains unchanged (exactly as you last changed it) and you have filename.ext.r* files. If you want to discard your changes and keep the repository version, just use the Revert command. If you want to keep your version and overwrite the repository version, use the Resolved command, then commit your version.

## Property conflicts

A property conflict occurs when two or more developers have changed the same property.
Example:
Develop A changes properties and commit first.

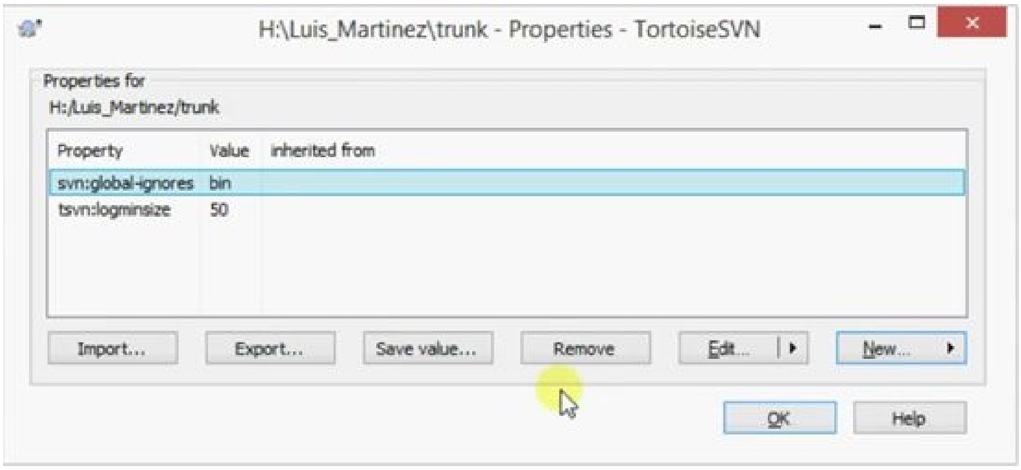


H:\svnTest2\tools - Properties - TortoiseSVN

Properties for
H:/svnTest2/tools

| Property | Value | inherited from |
|---|---|---|
| tsvn:logminsize | 100 | |

Developer B, try to commit after.

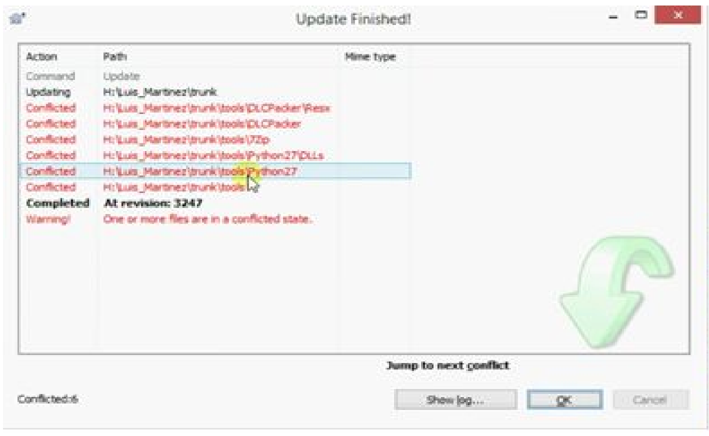


H:\Luis_Martinez\trunk - Properties - TortoiseSVN

Properties for
H:/Luis_Martinez/trunk

| Property | Value | inherited from |
|---|---|---|
| svn:global-ignores | bin | |
| tsvn:logminsize | 50 | |

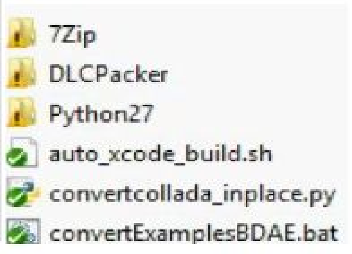
Then on update you will get:



Update Finished!

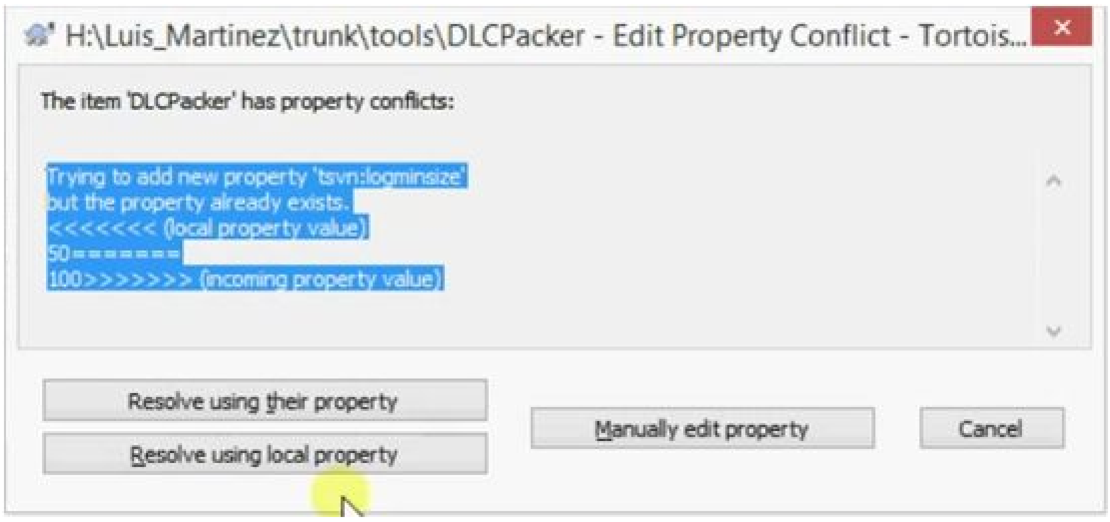
```
Action      Path                                               Mime type
Command     Update
Updating    H:\Luis_Martinez\trunk
Conflicted  H:\Luis_Martinez\trunk\tools\DLCPacker\Resx
Conflicted  H:\Luis_Martinez\trunk\tools\DLCPacker
Conflicted  H:\Luis_Martinez\trunk\tools\7Zip
Conflicted  H:\Luis_Martinez\trunk\tools\Python27\DLLs
Conflicted  H:\Luis_Martinez\trunk\tools\Python27
Conflicted  H:\Luis_Martinez\trunk\tools
Completed   At revision: 3247
Warning!    One or more files are in a conflicted state.
```

Conflicted:6



**How to resolve:**

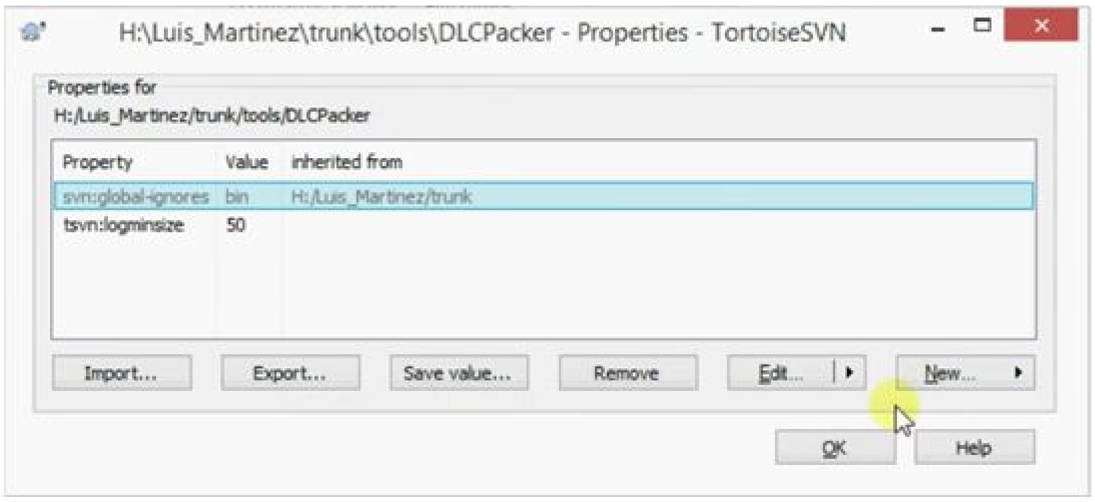
As with file content, resolving the conflict can only be done by the developers.
If the changes must be merged then select **Manually edit property,** sort out what the property value should be and mark as resolved.
Right click and select **TortoiseSVN** → **Edit Conflicts**, the below windows will be showed:


H:\Luis_Martinez\trunk\tools\DLCPacker - Edit Property Conflict - Tortois...
Resolve using their property
Resolve using local property
Manually edit property
Cancel

The item 'DLCPacker' has property conflicts:

Trying to add new property 'tsvn:logminsize'
but the property already exists.
<<<<<<< (local property value)
50=======
100>>>>>>> (incoming property value)

if one of the changes must override the other then choose the option to **Resolve using local property** or **Resolve using remote property**.
If the changes must be merged then select **Manually edit property,** sort out what the property value should be and mark as resolved.
Manually edit property option


H:\Luis_Martinez\trunk\tools\DLCPacker - Properties - TortoiseSVN
Import...
Export...
Save value...
Remove
Edit...
New...
OK
Help

Properties for
H:/Luis_Martinez/trunk/tools/DLCPacker

| Property | Value | inherited from |
|---|---|---|
| svn:global-ignores | bin | H:/Luis_Martinez/trunk |
| tsvn:logminsize | 50 | |

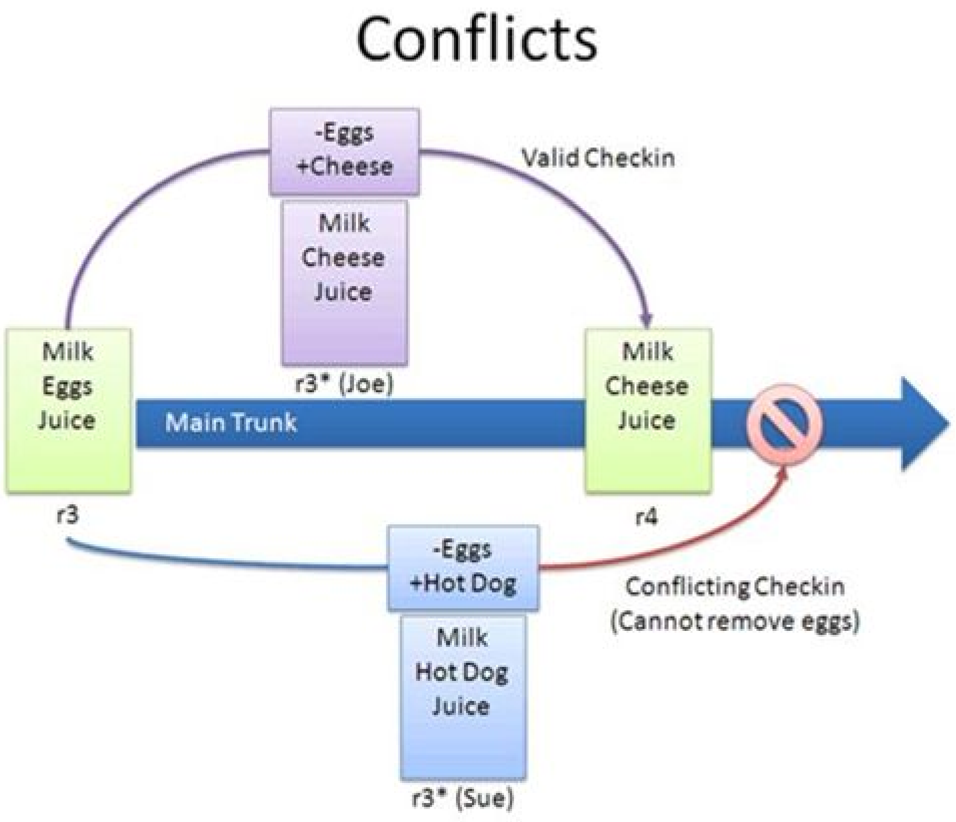
## Tree conflicts:

A tree conflict occurs when a developer moved/renamed/deleted a file or folder, which another developer either also has moved/renamed/deleted or just modified.

A tree conflict occurs when a developer moved/renamed/deleted a file or folder, which another developer either also has moved/renamed/deleted or just modified. There are many different situations that can result in a tree conflict, and all of them require different steps to resolve the conflict.
The tree conflict types are:

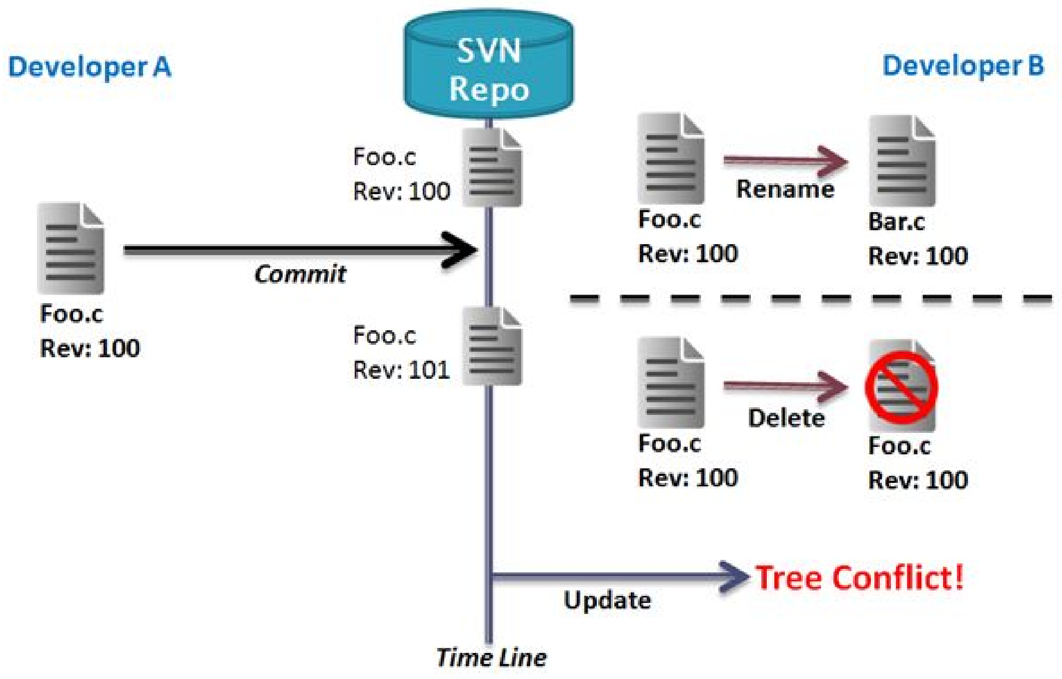
- Local delete, incoming edit upon update
- Local edit, incoming delete upon update
- Local delete, incoming delete upon update

- Local missing, incoming edit upon merge
- Local edit, incoming delete upon merge
- Local delete, incoming delete upon merge



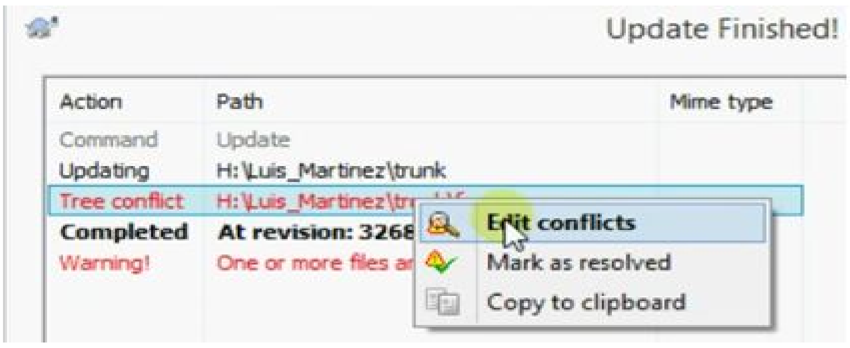
**<u>Note</u>**: TortoiseSVN can help find the right place to merge changes, but there may be additional work required to sort out the conflicts. Remember that after an update the working BASE will always contain the revision of each item as it was in the repository at the time of update. If you revert a change after updating it goes back to the repository state, not to the way it was when you started making your own local changes.

### Local delete, incoming edit upon update



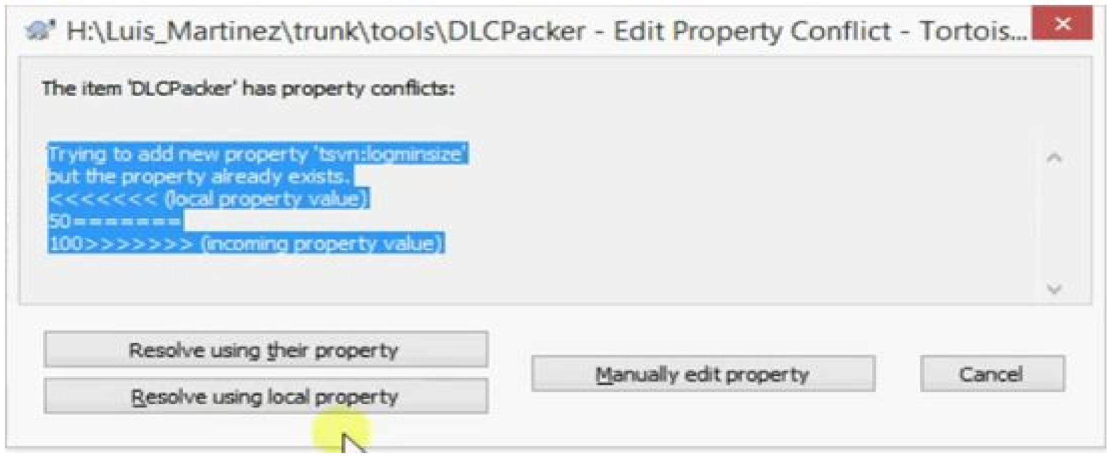
1. Developer A modifies Foo.c and commits it to the repository.
2. Developer B has simultaneously moved Foo.c to Bar.c in his working copy, or simply deleted Foo.c or its parent folder.
3. An update of developer B's working copy results in a tree conflict:
4. Foo.c has been deleted from working copy, but is marked with a tree conflict.
5. If the conflict results from a rename rather than a delete then Bar.c is marked as added, but does not contain developer A's modifications.

**How to resolve:**


Update Finished!

| Action | Path | Mime type |
|---|---|---|
| Command | Update | |
| Updating | H:\Luis_Martinez\trunk | |
| Tree conflict | H:\Luis_Martinez\tr | |
| Completed | At revision: 3268 | |
| Warning! | One or more files ar | |

Developer B now has to choose whether to keep Developer A's changes. In the case of a file rename, he can merge the changes to Foo.c into the renamed file Bar.c. For simple file or directory deletions he can choose to keep the item with Developer A's changes and discard the deletion. Or, by marking the conflict as resolved without doing anything he effectively discards Developer A's changes.

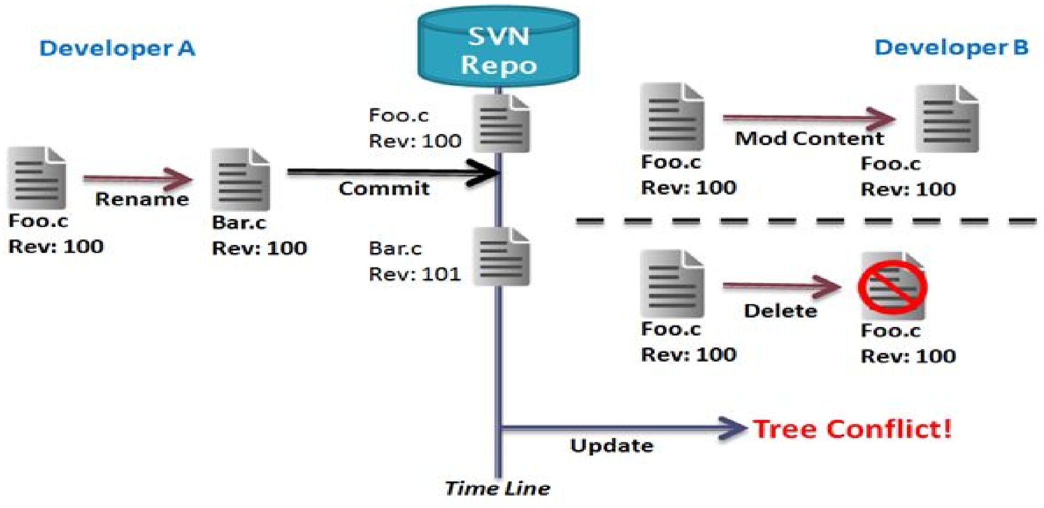

H:\Luis_Martinez\trunk\tools\DLCPacker - Edit Property Conflict - Tortois...

The item 'DLCPacker' has property conflicts:

```
Trying to add new property 'tsvn:logminsize'
but the property already exists.
<<<<<<< (local property value)
50=======
100>>>>>>> (incoming property value)
```

The conflict edit dialog offers to merge changes if it can find the original file of the renamed Bar.c. Depending on where the update was invoked, it may not be possible to find the source file.

### Local edit, incoming delete upon update



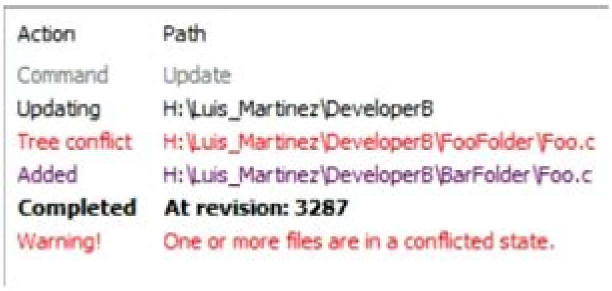
1. Developer A renames Foo.c to Bar.c and commits it to the repository.
2. Developer B modifies Foo.c in his working copy.

Or in the case of a folder move ...

1. Developer A moves parent folder FooFolder to BarFolder and commits it to the repository.
2. Developer B modifies Foo.c in his working copy.

An update of developer B's working copy results in a tree conflict. For a simple file conflict:

- Bar.c is added to the working copy as a normal file.
- Foo.c is marked as added (with history) and has a tree conflict.

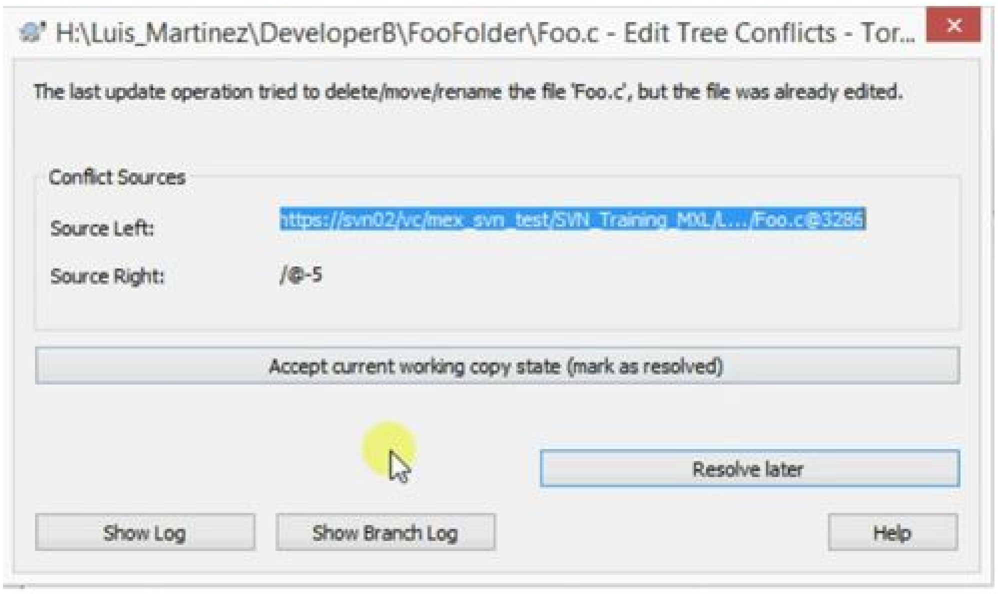


| Action | Path |
|---|---|
| Command | Update |
| Updating | H:\Luis_Martinez\DeveloperB |
| Tree conflict | H:\Luis_Martinez\DeveloperB\FooFolder\Foo.c |
| Added | H:\Luis_Martinez\DeveloperB\BarFolder\Foo.c |
| **Completed** | **At revision: 3287** |
| Warning! | One or more files are in a conflicted state. |

For a folder conflict:

- BarFolder is added to the working copy as a normal folder.
- FooFolder is marked as added (with history) and has a tree conflict.
- Foo.c is marked as modified.

**How to resolve:**



H:\Luis_Martinez\DeveloperB\FooFolder\Foo.c - Edit Tree Conflicts - Tor...

The last update operation tried to delete/move/rename the file 'Foo.c', but the file was already edited.

Conflict Sources

Source Left: https://svn02/vc/mex_svn_test/SVN_Training_MXL/L.../Foo.c@3286

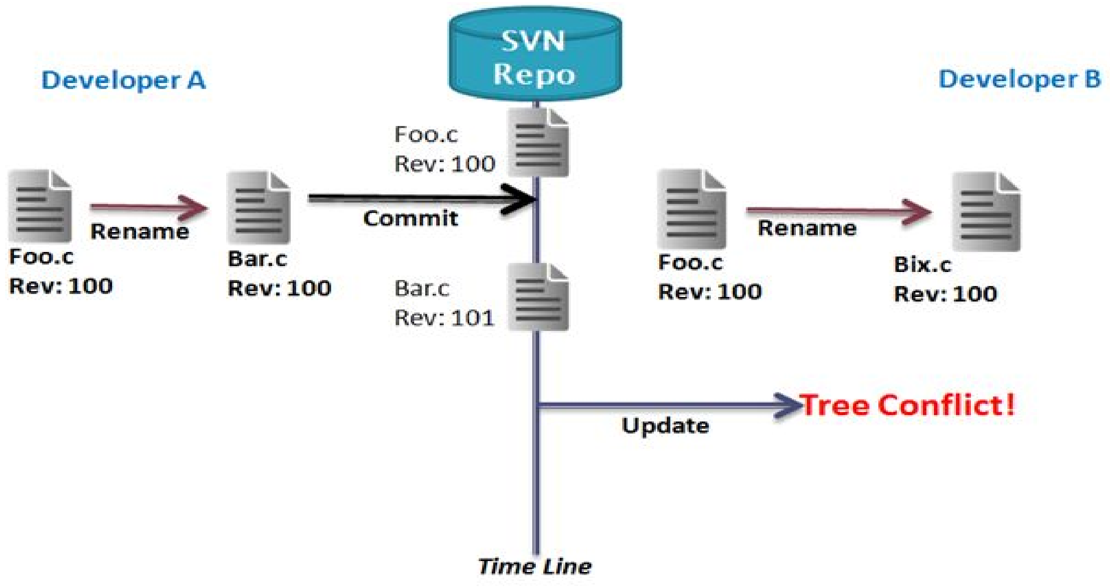
Source Right: /@-5

Developer B now has to decide whether to go with developer A's reorganization and merge her changes into the corresponding file in the new structure, or simply revert A's changes and keep the local file.

To merge her local changes with the reshuffle, Developer B must first find out to what filename the conflicted file Foo.c was renamed/moved in the repository. This can be done by using the log dialog. The changes must then be merged by hand as there is currently no way to automate or even simplify this process. Once the changes have been ported across, the conflicted path is redundant and can be deleted. In this case use the Remove button in the conflict editor dialog to clean up and mark the conflict as resolved.

If Developer B decides that A's changes were wrong then she must choose the Keep button in the conflict editor dialog. This marks the conflicted file/folder as resolved, but Developer A's changes need to be removed by hand. Again the log dialog helps to track down what was moved.

**Local delete, incoming delete upon update:**



1. Developer A renames Foo.c to Bar.c and commits it to the repository.
2. Developer B renames Foo.c to Bix.c.

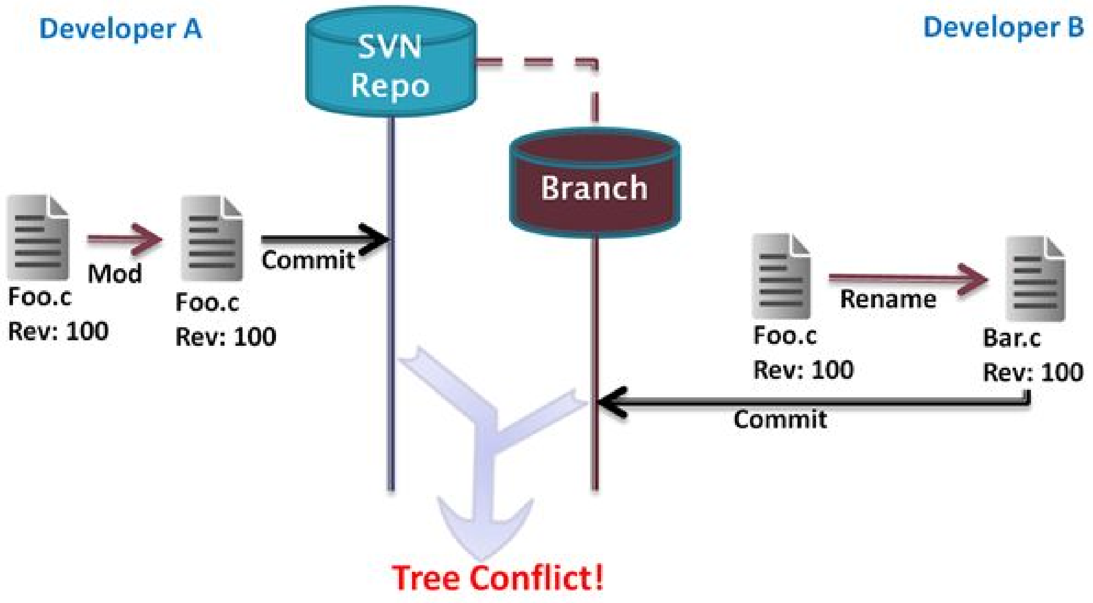
An update of developer B's working copy results in a tree conflict:

- Bix.c is marked as added with history.
- Bar.c is added to the working copy with status 'normal'.
- Foo.c is marked as deleted and has a tree conflict.

### How to resolve:

To resolve this conflict, Developer B has to find out to what filename the conflicted file Foo.c was renamed/moved in the repository. This can be done by using the log dialog.
Then developer B has to decide which new filename of Foo.c to keep - the one done by developer A or the rename done by himself.
After developer B has manually resolved the conflict, the tree conflict has to be marked as resolved with the button in the conflict editor dialog.

## 2.4 Local missing, incoming edit upon merge:



1. Developer A working on trunk modifies Foo.c and commits it to the repository
2. Developer B working on a branch moves Foo.c to Bar.c and commits it to the repository

A merge of developer A's trunk changes to developer B's branch working copy results in a tree conflict:

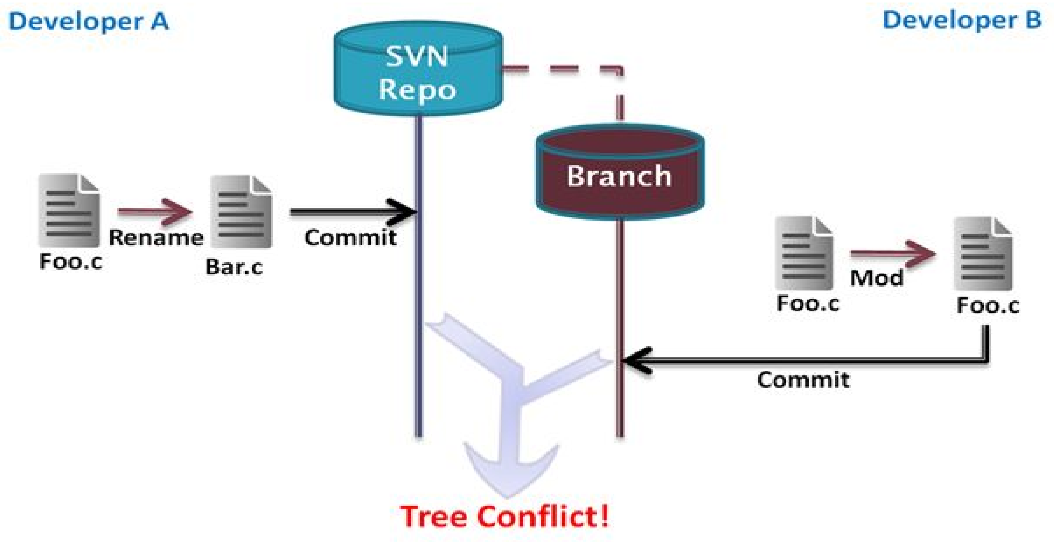
- Bar.c is already in the working copy with status 'normal'.
- Foo.c is marked as missing with a tree conflict.

### How to resolve:

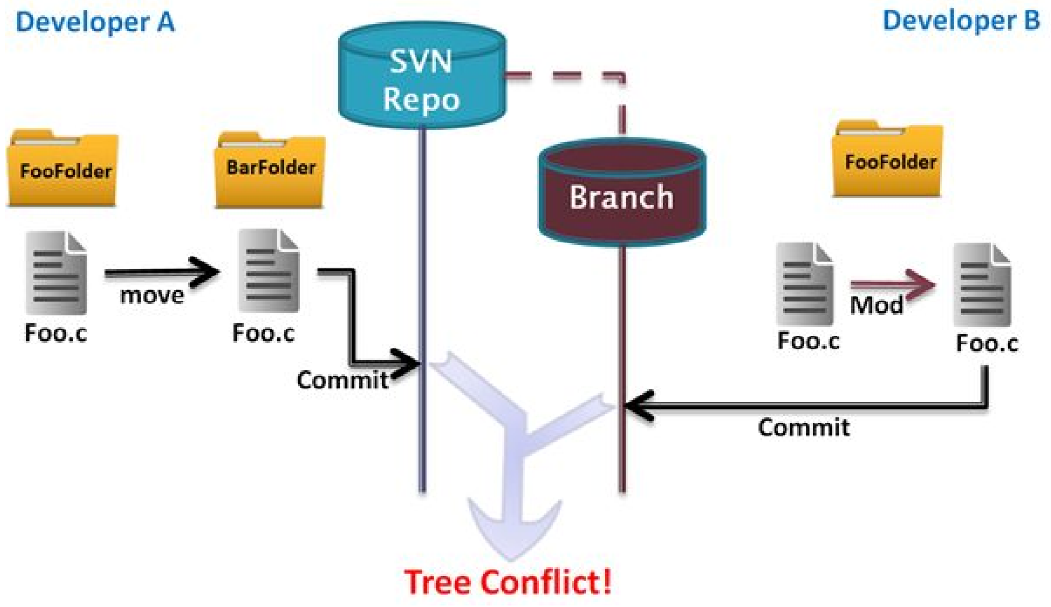
To resolve this conflict, Developer B has to mark the file as resolved in the conflict editor dialog, which will remove it from the conflict list. She then has to decide whether to copy the missing file Foo.c from the repository to the working copy, whether to merge Developer A's changes to Foo.c into the renamed Bar.c or whether to ignore the changes by marking the conflict as resolved and doing nothing else.

Note that if you copy the missing file from the repository and then mark as resolved, your copy will be removed again. You have to resolve the conflict first.

### Local edit, incoming delete upon merge



1. Developer A working on trunk moves Foo.c to Bar.c and commits it to the repository.
2. Developer B working on a branch modifies Foo.c and commits it to the repository.



1. Developer A working on trunk moves parent folder FooFolder to BarFolder and commits it to the repository.
2. Developer B working on a branch modifies Foo.c in her working copy.

A merge of developer A's trunk changes to developer B's branch working copy results in a tree conflict:

- Bar.c is marked as added.
- Foo.c is marked as modified with a tree conflict.

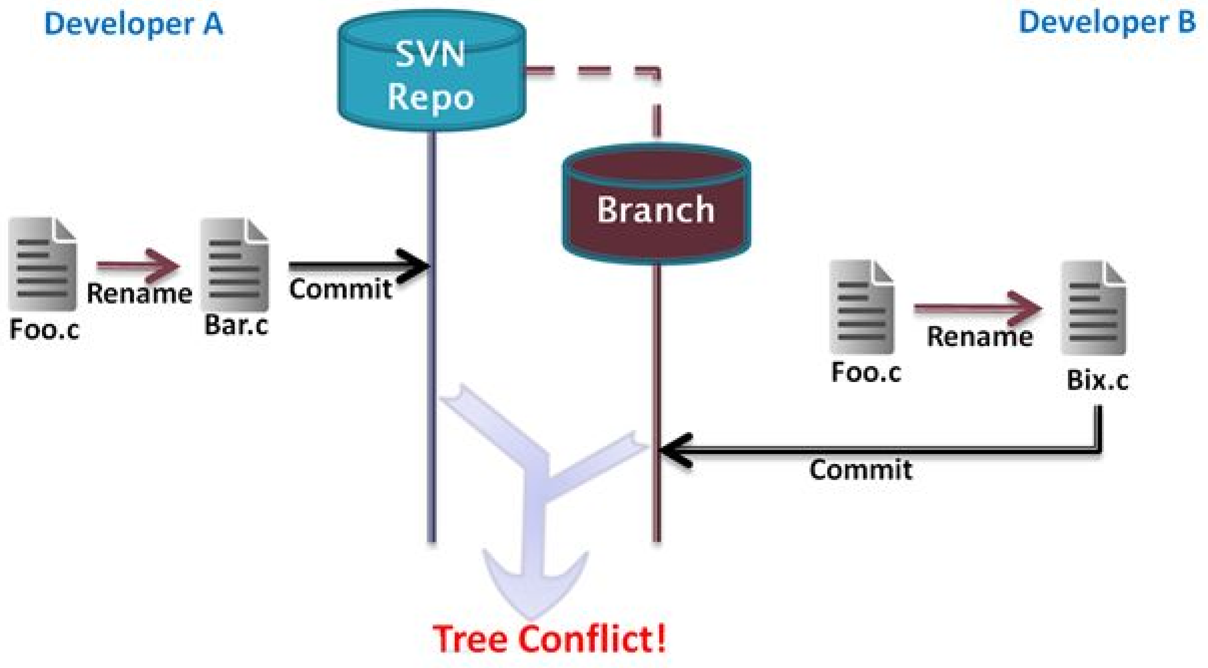
## How to resolve:

Developer B now has to decide whether to go with developer A's reorganization and merge her changes into the corresponding file in the new structure, or simply revert A's changes and keep the local file.

To merge her local changes with the reshuffle, Developer B must first find out to what filename the conflicted file Foo.c was renamed/moved in the repository. This can be done by using the log dialog for the merge source. The conflict editor only shows the log for the working copy as it does not know which path was used in the merge, so you will have to find that yourself. The changes must then be merged by hand as there is currently no way to automate or even simplify this process. Once the changes have been ported across, the conflicted path is redundant and can be deleted. In this case use the Remove button in the conflict editor dialog to clean up and mark the conflict as resolved.

If Developer B decides that A's changes were wrong then she must choose the Keep button in the conflict editor dialog. This marks the conflicted file/folder as resolved, but Developer A's changes need to be removed by hand. Again the log dialog for the merge source helps to track down what was moved.

### Local delete, incoming delete upon merge



1. Developer A working on trunk moves Foo.c to Bar.c and commits it to the repository.
2. Developer B working on a branch moves Foo.c to Bix.c and commits it to the repository.

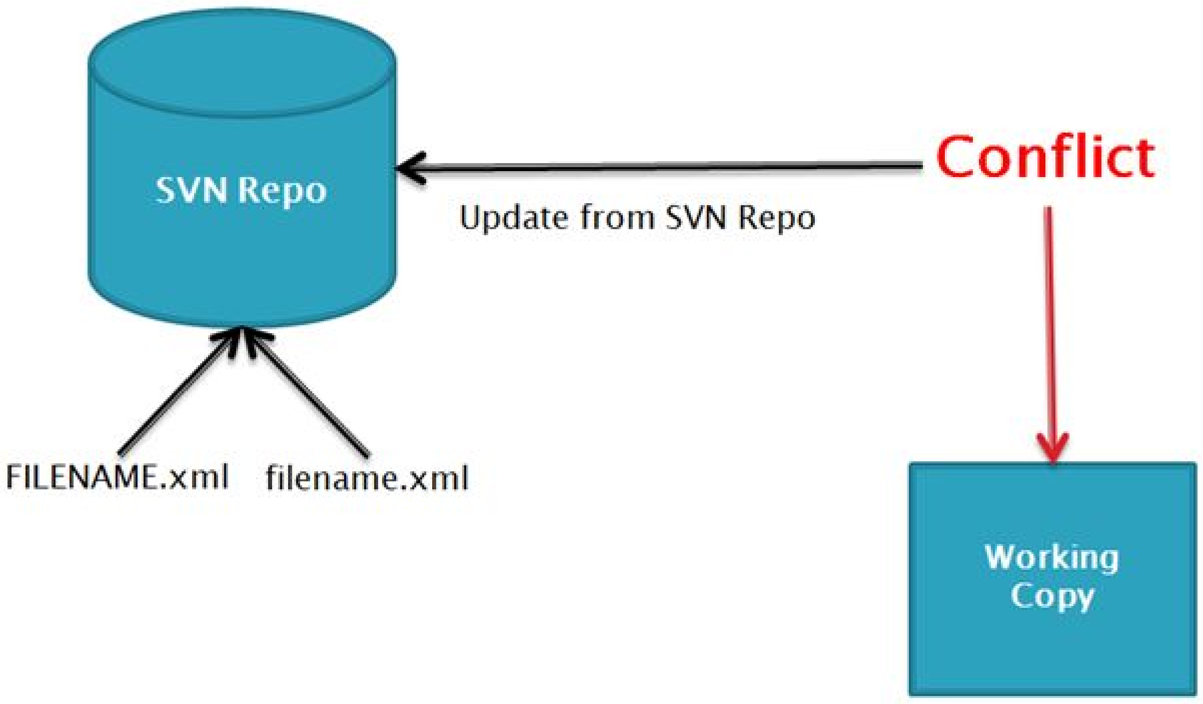
A merge of developer A's trunk changes to developer B's branch working copy results in a tree conflict:

- Bix.c is marked with normal (unmodified) status.
- Bar.c is marked as added with history.
- Foo.c is marked as missing and has a tree conflict.

### How to resolve:

To resolve this conflict, Developer B has to find out to what filename the conflicted file Foo.c was renamed/moved in the repository. This can be done by using the log dialog for the merge source. The conflict editor only shows the log for the working copy as it does not know which path was used in the merge, so you will have to find that yourself.
Then developer B has to decide which new filename of Foo.c to keep - the one done by developer A or the rename done by himself.
After developer B has manually resolved the conflict, the tree conflict has to be marked as resolved with the button in the conflict editor dialog.

## Upper/Lower-case file name conflicts

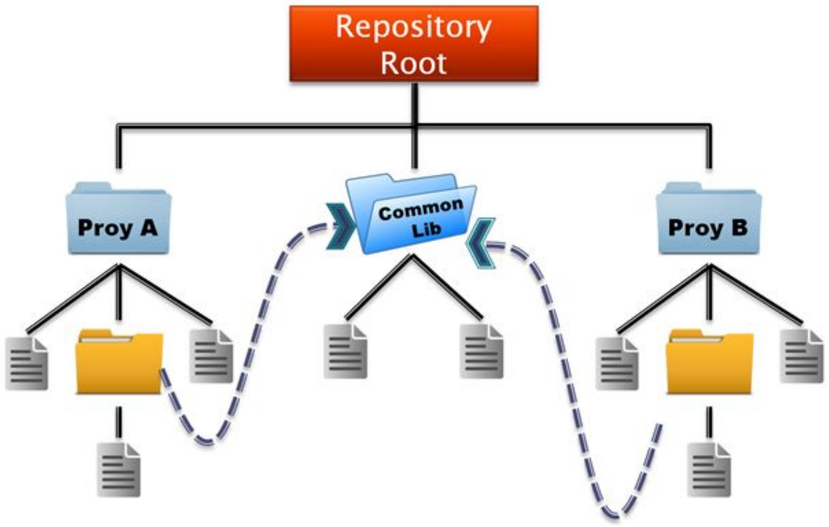


### How to resolve:

The easiest way would be to remove the file directly on the repository.

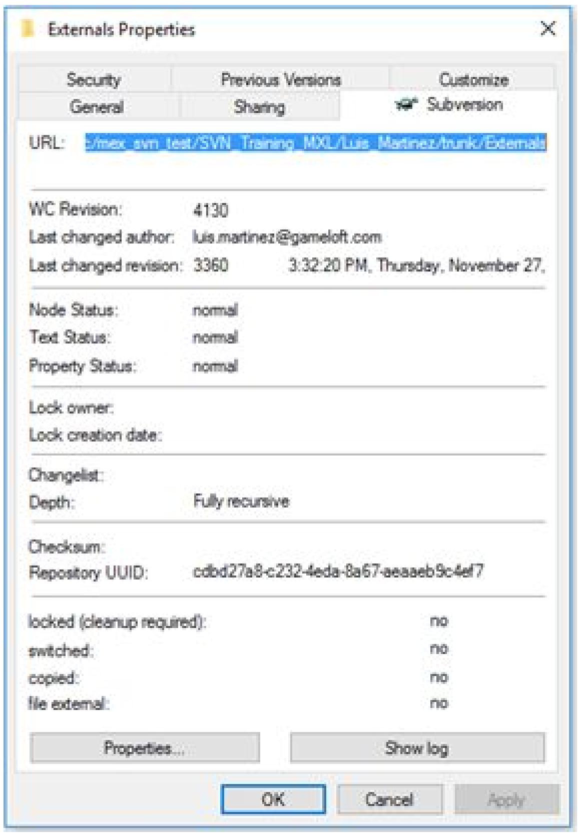
## Externals

Sometimes it is useful to construct a working copy that is made out of a number of different checkouts. For example, you may want different files or subdirectories to come from different locations in a repository, or perhaps from different repositories altogether, you can define the svn:externals properties to pull in the specified resource at the locations where they are needed.
In this example "Common Lib" have common tools for various projects, so each project can make an external from this folder. So you can do externals as needed to share common stuff, also can be locked to a specific revision number.



### Add an external folder to your working copy

**1. Right click** on your working copy folder where you want the external and choose **Windows Menu** → **Properties** from the context menu. The Properties Dialog comes up.
2. Then go to the **Subversion** tab. There, you can set properties.



URL: :/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk/Externals

WC Revision: 4130
Last changed author: luis.martinez@gameloft.com
Last changed revision: 3360 3:32:20 PM, Thursday, November 27,

Node Status: normal
Text Status: normal
Property Status: normal

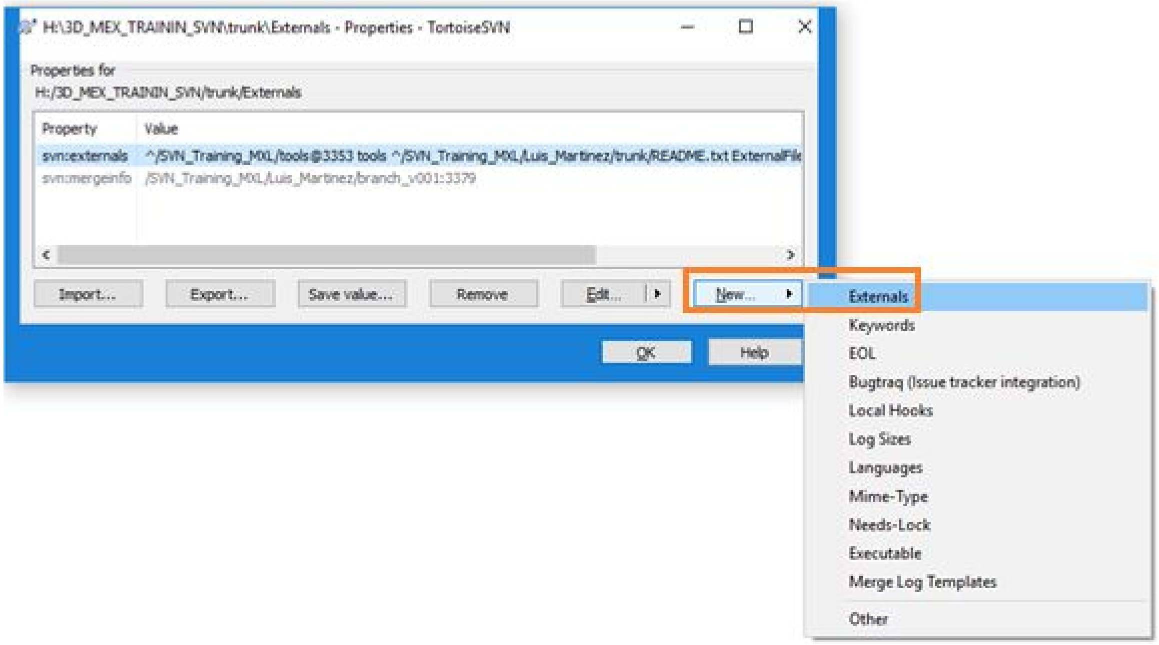
Lock owner:
Lock creation date:

Changelist:
Depth: Fully recursive

Checksum:
Repository UUID: cdbd27a8-c232-4eda-8a67-aeaaeb9c4ef7

locked (cleanup required): no
switched: no
copied: no
file external: no

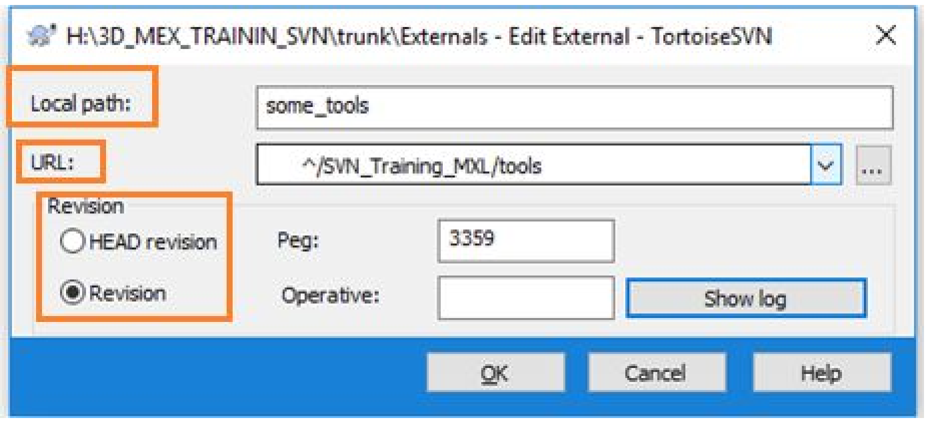
3. Click **Properties....** In the properties dialog, either double click on the svn:externals if it already exists, or click on the **New...**button and select externals from the menu.


H:\3D_MEX_TRAININ_SVN\trunk\Externals - Properties - TortoiseSVN

Properties for
H:/3D_MEX_TRAININ_SVN/trunk/Externals

| Property | Value |
|---|---|
| svn:externals | ^/SVN_Training_MXL/tools@3353 tools ^/SVN_Training_MXL/Luis_Martinez/trunk/README.txt ExternalFile |
| svn:mergeinfo | /SVN_Training_MXL/Luis_Martinez/branch_v001:3379 |

4. To add a new external, click the **New...** and then fill in the required information in the shown dialog.


H:\3D_MEX_TRAININ_SVN\trunk\Externals - Edit External - TortoiseSVN

Local path: some_tools

URL: ^/SVN_Training_MXL/tools

Revision: HEAD revision [✓ Revision]

Peg: 3359

Operative:

**Local Path**

Refers to the file or folder relative to the folder that has this externals property set on it. ex. **Local Path:** Externals *(note that it must not exist in the current directory; it will be created).*

**URL**

Is the item's URL you want to check out as external in your project. You can use relative or absolute URL. Also you can use the "three points" to open SVN browser and select the desired item to be external.

**Peg / Operative / HEAD revision**

**HEAD revision**, is the most recent revision on SVN, and when you update this external will be updated to the newest revision on SVN.

**Peg revision**, it is for set an external to an specific revision, and will stay in the same revision on update. This is the recommended configuration for your project.

**Operative revision**, to instruct Subversion to work with these reused paths. The simplest example to use operative revision is when a directory or file is deleted from version control, and then a new directory or file is created with the same name and added to version control. The thing you deleted and the thing you later added aren't the same thing. They merely happen to have had the same path—/trunk/object, for example. What, then, does it mean to ask Subversion about the history of /trunk/object? Are you asking about the thing currently at that location, or the old thing you deleted from that location? Are you asking about the operations that have happened to *all* the objects that have ever lived at that path? Subversion needs a hint about what you really want.

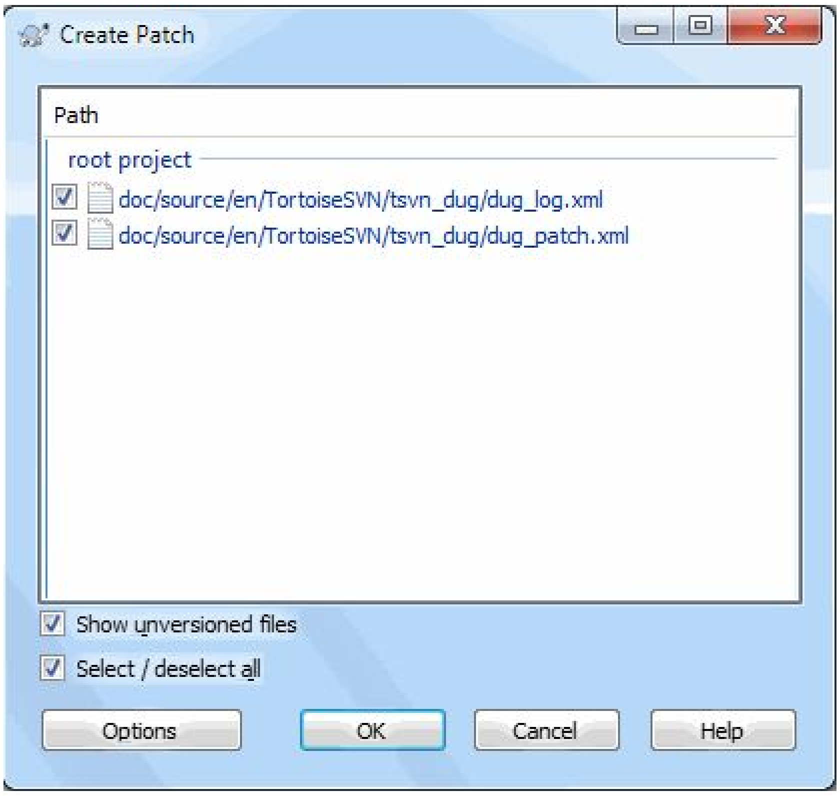
5. After fill all the information needed, click ok on all windows showed to confirm changes.
6. Next update your working copy using right click and select **SVN Update** on context menu.
7. Finally commit your working copy, the external is property in your working copy, and must be committed.

# Creating and applying Patches

For Gameloft projects, every member of a team has read access to the repository and make a contribution to the project of team. So how are those contributions controlled? If just anyone could commit changes, the project would be permanently unstable and probably permanently broken. In this situation the change is managed by submitting a *patch* file to the development team, who do have write access. They can review the patch first, and then either submit it to the repository or reject it back to the author.
Patch files are simply Unified-Diff files showing the differences between your working copy and the base revision.

## Creating a Patch File

First you need to make *and test* your changes. Then instead of using TortoiseSVN → Commit... on the parent folder, you select TortoiseSVN → Create Patch...



☑ doc/source/en/TortoiseSVN/tsvn_dug/dug_log.xml
☑ doc/source/en/TortoiseSVN/tsvn_dug/dug_patch.xml

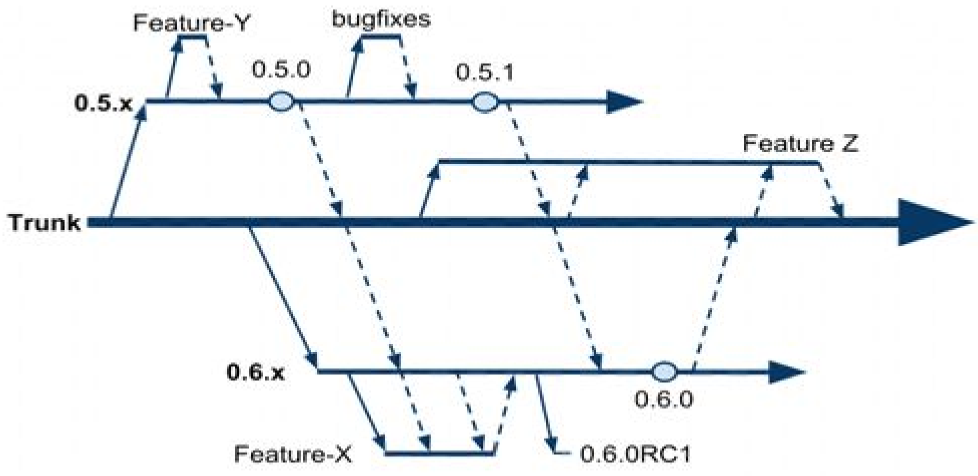
You can now select the files you want included in the patch, just as you would with a full commit. This will produce a single file containing a summary of all the changes you have made to the selected files since the last update from the repository.
By clicking on the **Options** button you can specify how the patch is created. For example you can specify that changes in line endings or white spaces are not included in the final patch file.

You can produce separate patches containing changes to different sets of files. Of course, if you create a patch file, make some more changes to the *same* files and then create another patch, the second patch file will include *both* sets of changes.
Just save the file using a filename of your choice. Patch files can have any extension you like, but by convention they should use the .patcher .diff extension. You are now ready to submit your patch file.
You can also save the patch to the clipboard instead of to a file. You might want to do this so that you can paste it into an email for review by others. Or if you have two working copies on one machine and you want to transfer changes from one to the other, a patch on the clipboard is a convenient way of doing this.
If you prefer, you can create a patch file from within the Commit or Check for Modifications dialogs. Just select the files and use the context menu item to create a patch from those files. If you want to see the Options dialog you have to hold **shift** when you right click.

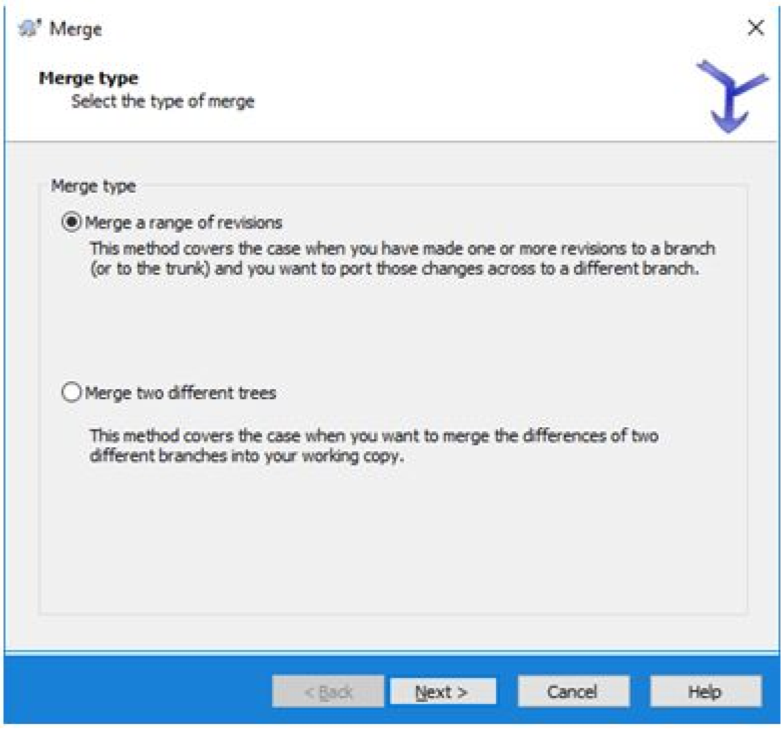
## Applying a Patch File

Patch files are applied to your working copy. This should be done from the same folder level as was used to create the patch. If you are not sure what this is, just look at the first line of the patch file. For example, if the first file being worked on wasdoc/source/english/chapter1.xml and the first line in the patch file is Index: english/chapter1.xml then you need to apply the patch to the doc/source/folder. However, provided you are in the correct working copy, if you pick the wrong folder level, TortoiseSVN will notice and suggest the correct level.
In order to apply a patch file to your working copy, you need to have at least read access to the repository. The reason for this is that the merge program must reference the changes back to the revision against which they were made by the remote developer.
From the context menu for that folder, click on TortoiseSVN → Apply Patch... This will bring up a file open dialog allowing you to select the patch file to apply. By default only .patch or .diff files are shown, but you can opt for “All files”. If you previously saved a patch to the clipboard, you can use Open from clipboard... in the file open dialog. Note that this option only appears if you saved the patch to the clipboard using TortoiseSVN → Create Patch.... Copying a patch to the clipboard from another app will not make the button appear.
Alternatively, if the patch file has a .patch or .diff extension, you can right click on it directly and select TortoiseSVN → Apply Patch.... In this case you will be prompted to enter a working copy location.
These two methods just offer different ways of doing the same thing. With the first method you select the WC and browse to the patch file. With the second you select the patch file and browse to the WC.
Once you have selected the patch file and working copy location, TortoiseMerge runs to merge the changes from the patch file with your working copy. A small window lists the files which have been changed. Double click on each one in turn, review the changes and save the merged files.
The remote developer's patch has now been applied to your working copy, so you need to commit to allow everyone else to access the changes from the repository.

## How to merge



Where branches are used to maintain separate lines of development, at some stage you will want to merge the changes made on one branch back into the trunk, or vice versa.
The merging *always* takes place within a working copy. If you want to merge changes *into* a branch, you have to have a working copy for that branch checked out, and invoke the merge wizard from that working copy using **TortoiseSVN** → **Merge** from context menu.



Merge type
Select the type of merge

Merge type

(●) Merge a range of revisions

This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

( ) Merge two different trees

This method covers the case when you want to merge the differences of two different branches into your working copy.

**<u>Note</u>**: In general it is a good idea to perform a merge into an unmodified working copy. If you have made other changes in your working copy, commit those first. If the merge does not go as you expect, you may want to revert the changes, and the Revert command will discard *all* changes including any you made before the merge.

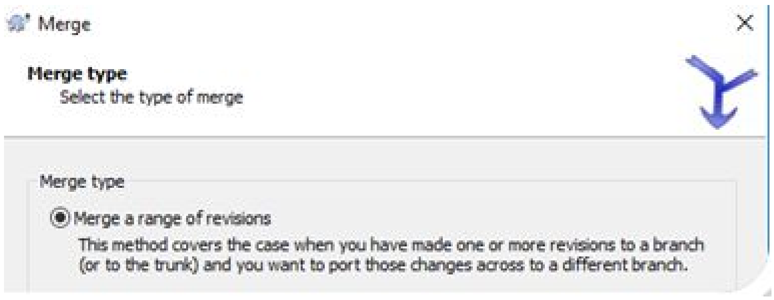
## Merge a range of revisions

This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

What you are asking Subversion to do is this: “ Calculate the changes necessary to get [FROM] revision 1 of branch A [TO] revision 7 of branch A, and apply those changes to my working copy (of trunk or branch B).”

How to:

1. Invoke the merge wizard from that working copy using **TortoiseSVN** → **Merge** from context menu.
2. Select "**Merge range of revisions**" option, and then click on **Next** button.

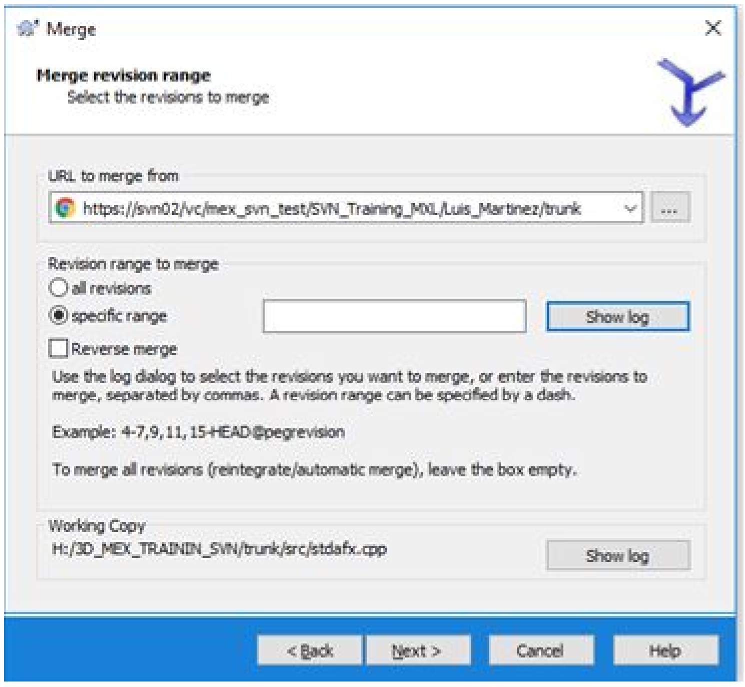


Merge

Merge type
Select the type of merge

Merge type
(●) Merge a range of revisions
This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

3. On the next window add the full folder URL of the branch or tag containing the changes you want to port into your working copy. You may also click ... to browse the repository and find the desired branch.



Merge

Merge revision range
Select the revisions to merge

URL to merge from
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk

Revision range to merge
( ) all revisions
(●) specific range
[ ] Reverse merge
Use the log dialog to select the revisions you want to merge, or enter the revisions to merge, separated by commas. A revision range can be specified by a dash.

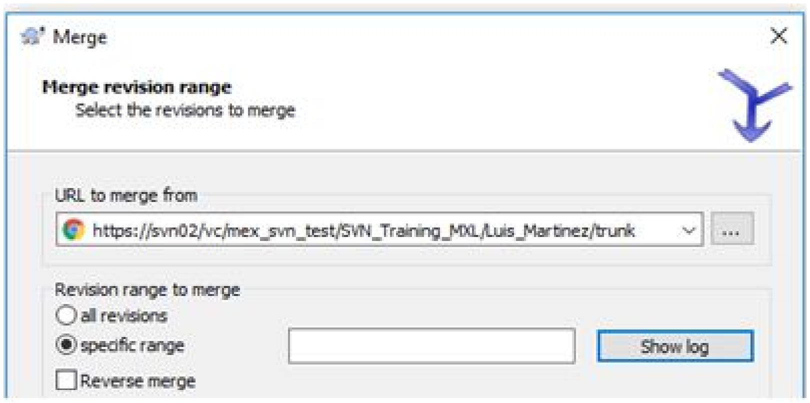
Example: 4-7,9,11,15-HEAD@pegrevision

To merge all revisions (reintegrate/automatic merge), leave the box empty.

Working Copy
H:/3D_MEX_TRAININ_SVN/trunk/src/stdafx.cpp

4. Next, if you leave the revision range empty, Subversion uses the merge-tracking features to calculate the correct revision range to use. This is known as a reintegrate or automatic merge.

In the Revision range to merge field enter the list of revisions you want to merge. This can be a single revision, a list of specific revisions separated by commas, or a range of revisions separated by a dash, or any combination of these. You can click on **Show log** button to select specific revisions on history log.



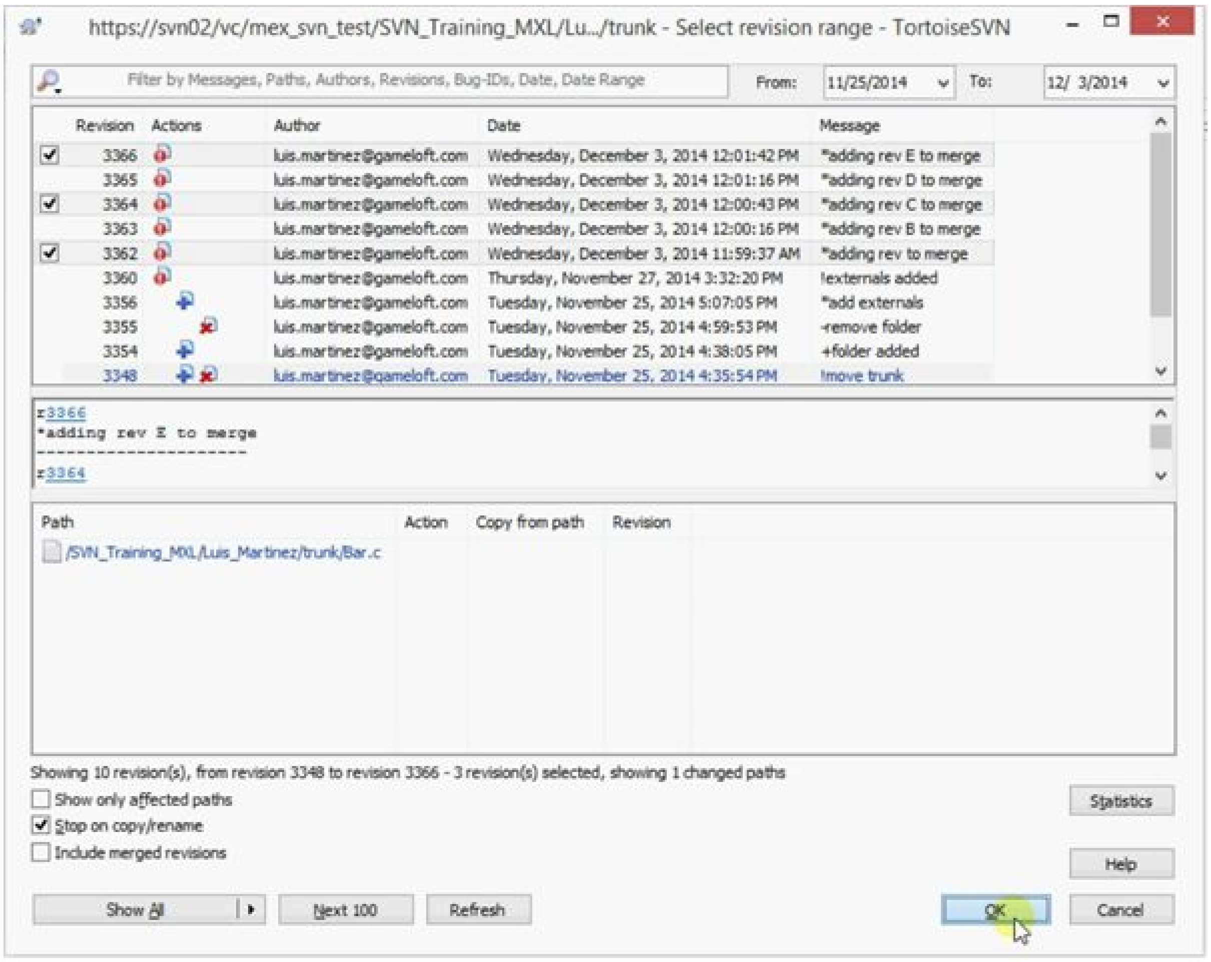
**Merge revision range**
Select the revisions to merge

URL to merge from
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk

Revision range to merge
( ) all revisions
(●) specific range

☐ Reverse merge

5. If you click on **Show Log**, the below windows will appear, and there you can select the desired revisions, then click on **Ok** button.


https://svn02/vc/mex_svn_test/SVN_Training_MXL/Lu.../trunk - Select revision range - TortoiseSVN

Filter by Messages, Paths, Authors, Revisions, Bug-IDs, Date, Date Range

From: 11/25/2014 To: 12/ 3/2014

| | Revision | Actions | Author | Date | Message |
|---|---|---|---|---|---|
| ✓ | 3366 | | luis.martinez@gameloft.com | Wednesday, December 3, 2014 12:01:42 PM | *adding rev E to merge |
| | 3365 | | luis.martinez@gameloft.com | Wednesday, December 3, 2014 12:01:16 PM | *adding rev D to merge |
| ✓ | 3364 | | luis.martinez@gameloft.com | Wednesday, December 3, 2014 12:00:43 PM | *adding rev C to merge |
| | 3363 | | luis.martinez@gameloft.com | Wednesday, December 3, 2014 12:00:16 PM | *adding rev B to merge |
| ✓ | 3362 | | luis.martinez@gameloft.com | Wednesday, December 3, 2014 11:59:37 AM | *adding rev to merge |
| | 3360 | | luis.martinez@gameloft.com | Thursday, November 27, 2014 3:32:20 PM | !externals added |
| | 3356 | | luis.martinez@gameloft.com | Tuesday, November 25, 2014 5:07:05 PM | *add externals |
| | 3355 | | luis.martinez@gameloft.com | Tuesday, November 25, 2014 4:59:53 PM | -remove folder |
| | 3354 | | luis.martinez@gameloft.com | Tuesday, November 25, 2014 4:38:05 PM | +folder added |
| | 3348 | | luis.martinez@gameloft.com | Tuesday, November 25, 2014 4:35:54 PM | !move trunk |

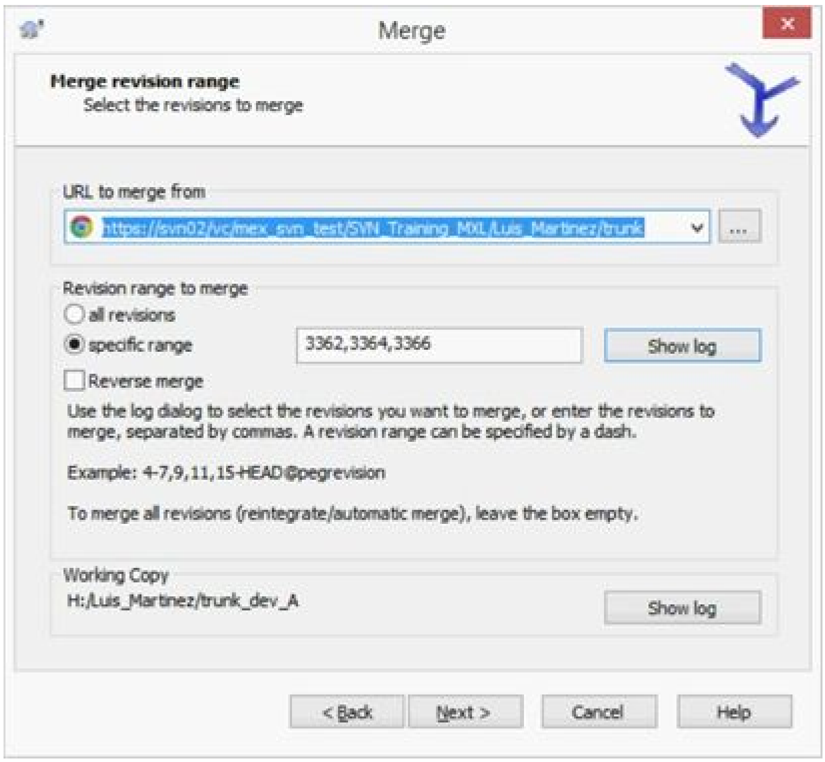
```
r3366
*adding rev E to merge
---------------------
r3364
```

| Path | Action | Copy from path | Revision |
|---|---|---|---|
| /SVN_Training_MXL/Luis_Martinez/trunk/Bar.c | | | |

Showing 10 revision(s), from revision 3348 to revision 3366 - 3 revision(s) selected, showing 1 changed paths

☐ Show only affected paths
☑ Stop on copy/rename
☐ Include merged revisions

6. Next you will be returned to the last windows, but now you will have the revision range filled, Then click on Next button.


Merge

**Merge revision range**
Select the revisions to merge

URL to merge from: https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk

Revision range to merge
- ○ all revisions
- ● specific range: 3362,3364,3366
- ☐ Reverse merge

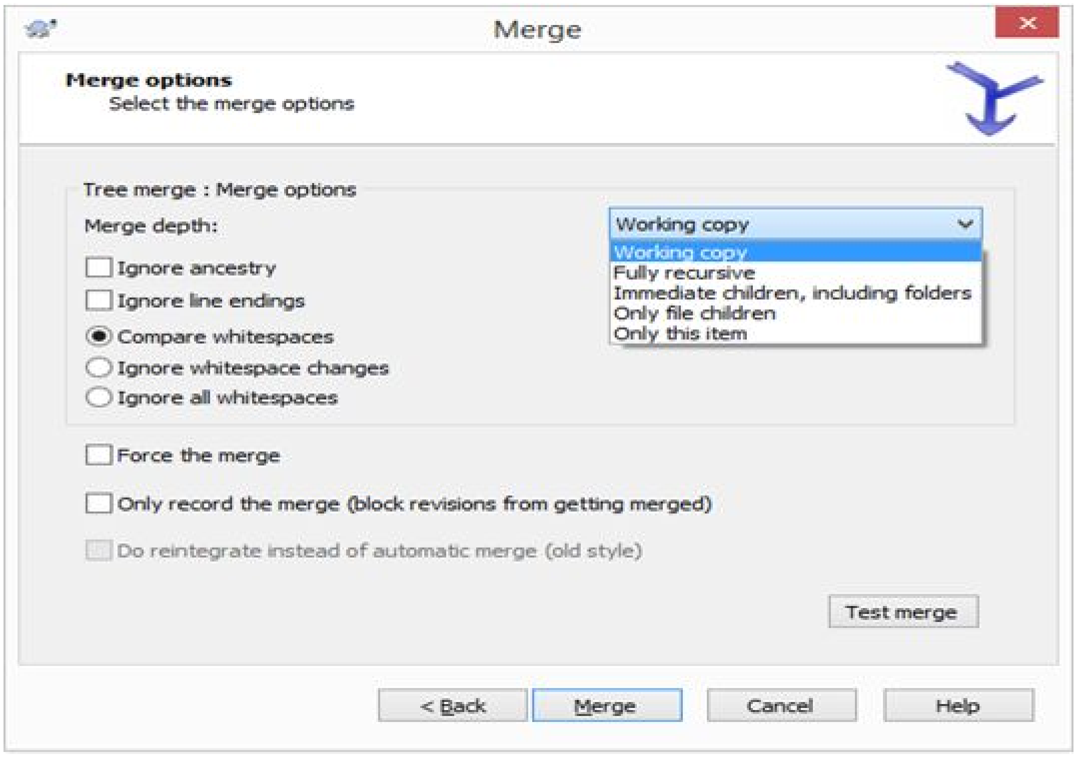
Use the log dialog to select the revisions you want to merge, or enter the revisions to merge, separated by commas. A revision range can be specified by a dash.

Example: 4-7,9,11,15-HEAD@pegrevision

To merge all revisions (reintegrate/automatic merge), leave the box empty.

Working Copy: H:/Luis_Martinez/trunk_dev_A

7. After, the below windows will appear, then click on **Merge**. This windows will be explaned below.


Merge

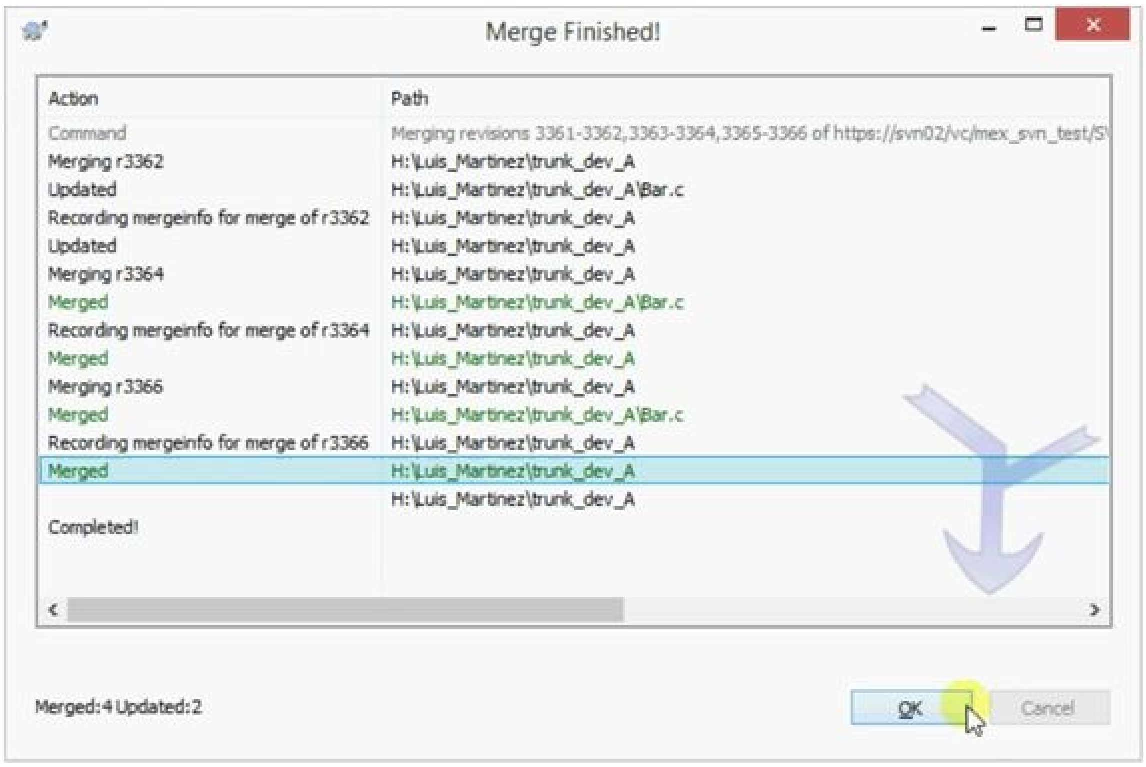
**Merge options**
Select the merge options

Tree merge : Merge options

Merge depth: Working copy

- ☐ Ignore ancestry
- ☐ Ignore line endings
- ● Compare whitespaces
- ○ Ignore whitespace changes
- ○ Ignore all whitespaces
- ☐ Force the merge
- ☐ Only record the merge (block revisions from getting merged)
- ☐ Do reintegrate instead of automatic merge (old style)

8. Finally the merge will be doing automatically, and status windows will be showed.



Merge Finished!

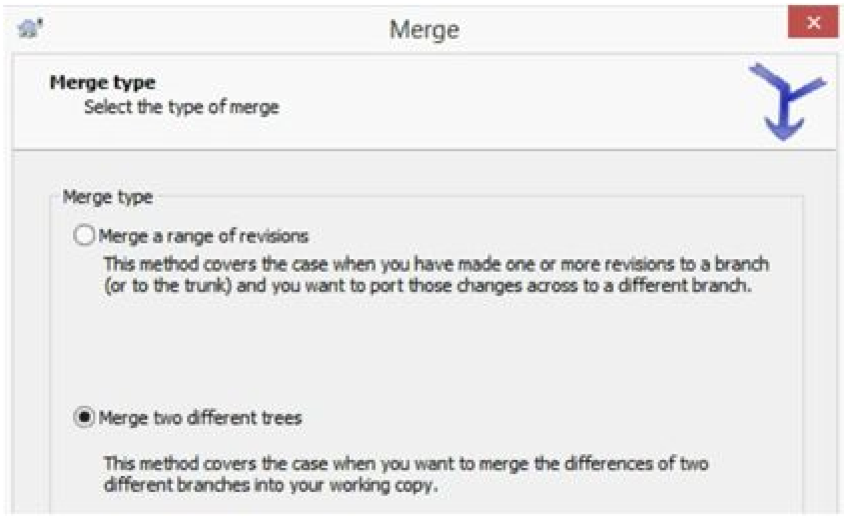
| Action | Path |
|---|---|
| Command | Merging revisions 3361-3362,3363-3364,3365-3366 of https://svn02/vc/mex_svn_test/S\ |
| Merging r3362 | H:\Luis_Martinez\trunk_dev_A |
| Updated | H:\Luis_Martinez\trunk_dev_A\Bar.c |
| Recording mergeinfo for merge of r3362 | H:\Luis_Martinez\trunk_dev_A |
| Updated | H:\Luis_Martinez\trunk_dev_A |
| Merging r3364 | H:\Luis_Martinez\trunk_dev_A |
| Merged | H:\Luis_Martinez\trunk_dev_A\Bar.c |
| Recording mergeinfo for merge of r3364 | H:\Luis_Martinez\trunk_dev_A |
| Merged | H:\Luis_Martinez\trunk_dev_A |
| Merging r3366 | H:\Luis_Martinez\trunk_dev_A |
| Merged | H:\Luis_Martinez\trunk_dev_A\Bar.c |
| Recording mergeinfo for merge of r3366 | H:\Luis_Martinez\trunk_dev_A |
| Merged | H:\Luis_Martinez\trunk_dev_A |
| | H:\Luis_Martinez\trunk_dev_A |
| Completed! | |

Merged:4 Updated:2

## Merging two different trees

This is a more general case of the reintegrate method. What you are asking Subversion to do is: " Calculate the changes necessary to get [FROM] the head revision of the trunk [TO] the head revision of the branch, and apply those changes to my working copy (of the trunk). " The next result is that trunk now looks exactly like the branch.

Note: If you are using this method to merge a feature branch back to trunk, you need to start the merge wizard from within a working copy of trunk.



Merge

**Merge type**
Select the type of merge

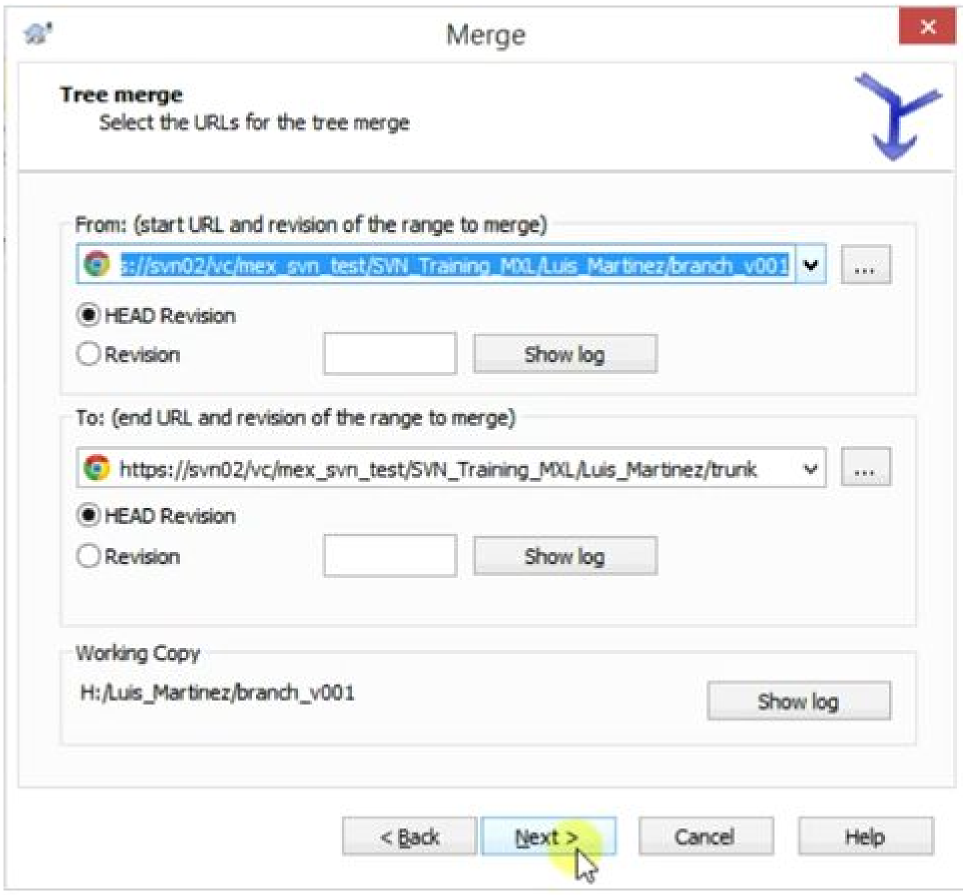
Merge type

( ) Merge a range of revisions
This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

(●) Merge two different trees
This method covers the case when you want to merge the differences of two different branches into your working copy.

How to:

After select Merge two diff. trees, a new windows winn appear and you need to set the below information, and click on **Next** button:



Merge

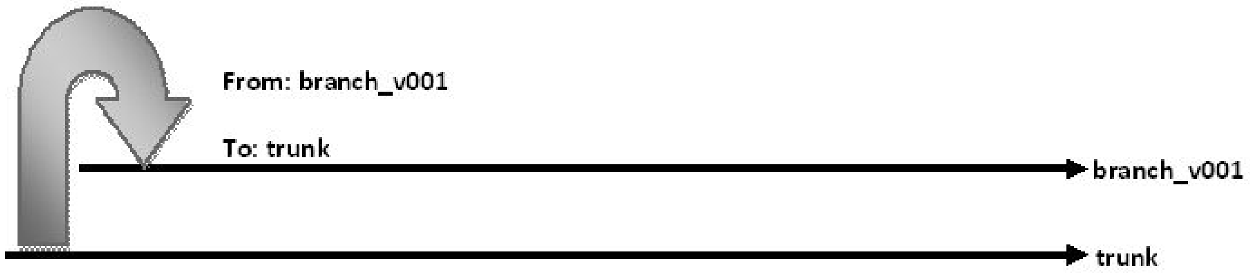
**Tree merge**
Select the URLs for the tree merge

From: (start URL and revision of the range to merge)
s://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/branch_v001
(●) HEAD Revision
( ) Revision

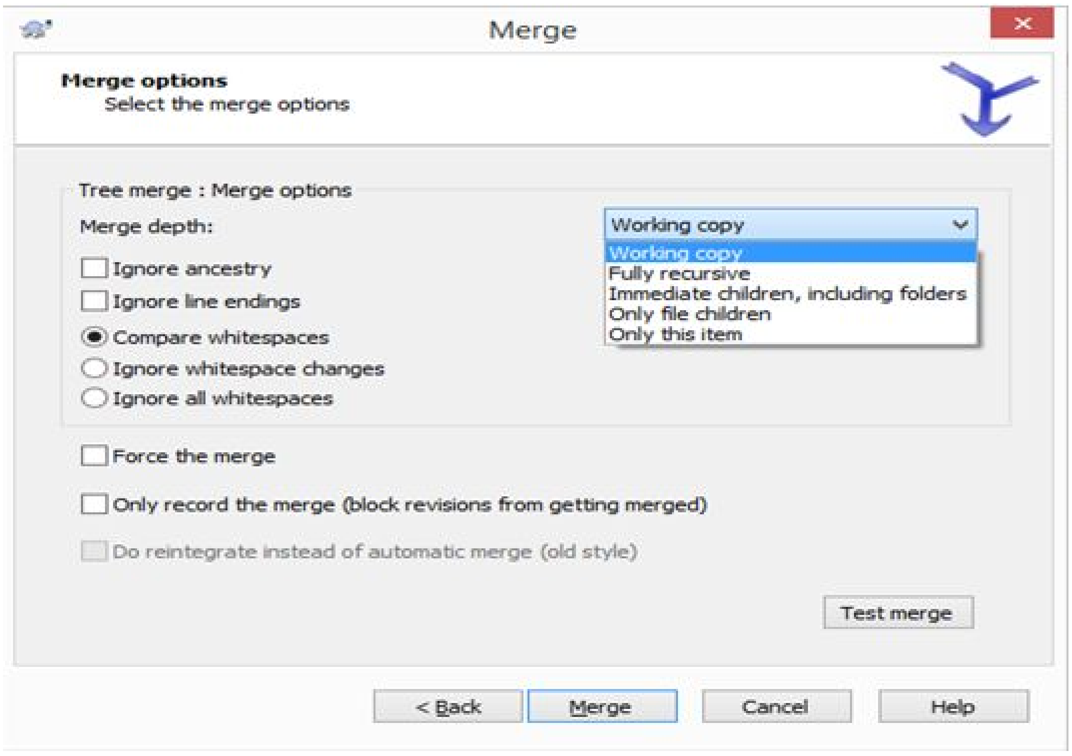
To: (end URL and revision of the range to merge)
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk
(●) HEAD Revision
( ) Revision

Working Copy
H:/Luis_Martinez/branch_v001

**From**: Enter in the field the full folder URL of the *trunk*. This may sound wrong, but remember that the trunk is the start point to which you want to add the branch changes. You may also click **...** to browse the repository.
**To**: Enter in the field the full folder URL of the feature branch.
**From Revision** / **To Revision**: Enter in the field the last revision number at which the two trees were synchronized. If you are sure no-one else is making commits you can use the HEAD revision in both cases. If there is a chance that someone else may have made a commit since that synchronization, use the specific revision number to avoid losing more recent commits. You can also use **Show Log** to select the revision.



1. Next, the below windows will appear, then click on Merge button. This windows will be explained below.



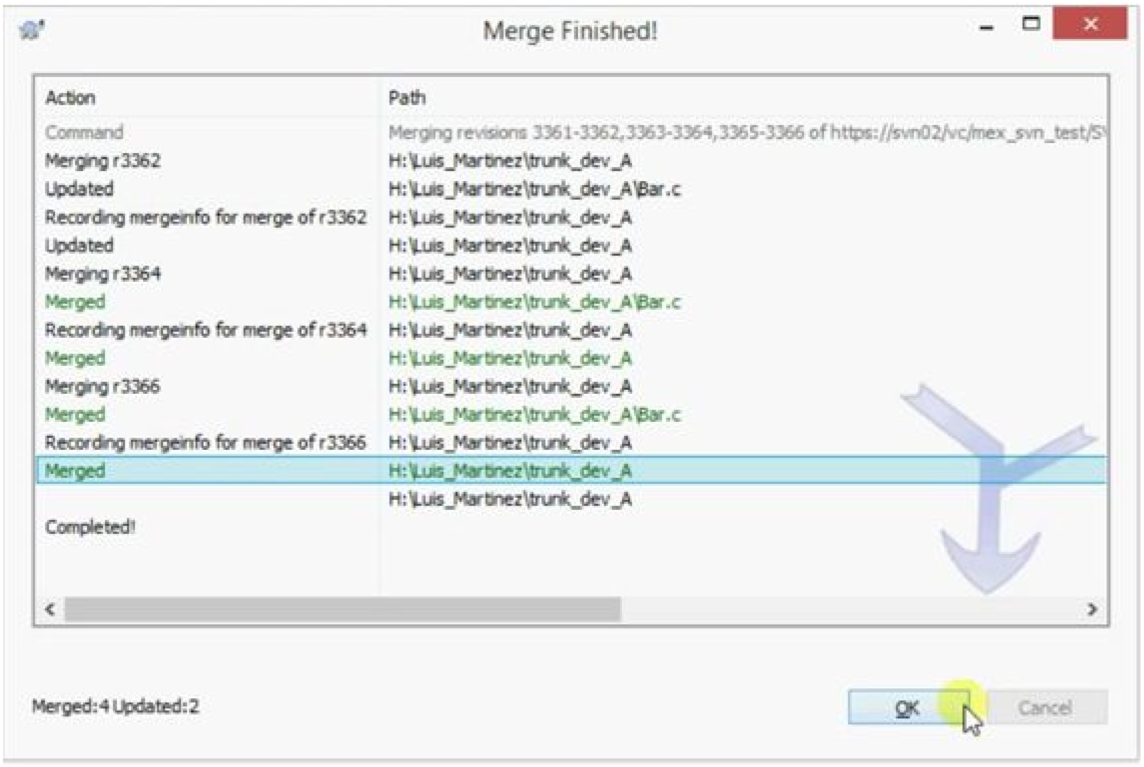
Tree merge : Merge options

Merge depth: Working copy

☐ Ignore ancestry
☐ Ignore line endings
● Compare whitespaces
○ Ignore whitespace changes
○ Ignore all whitespaces

☐ Force the merge
☐ Only record the merge (block revisions from getting merged)
☐ Do reintegrate instead of automatic merge (old style)

2. Finally the merge will be doing automatically, and status windows will be showed.



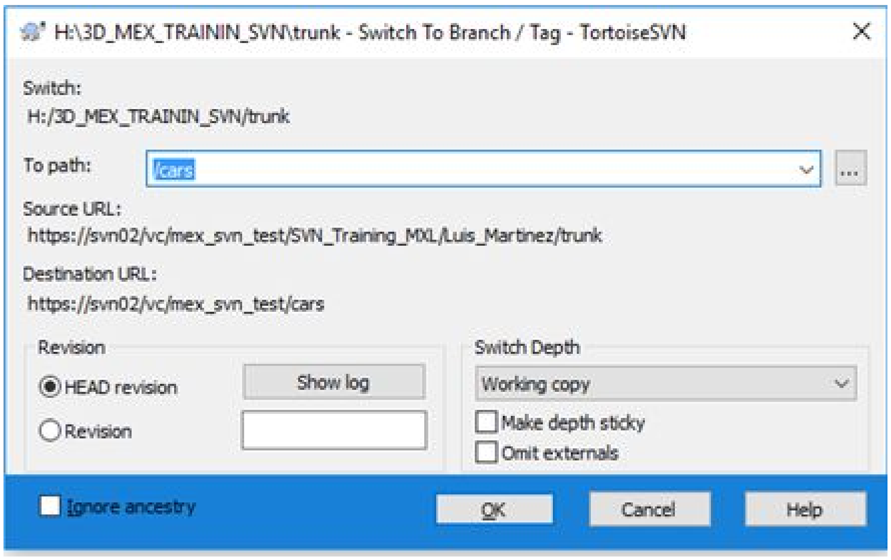
```
Action                              Path
Command                             Merging revisions 3361-3362,3363-3364,3365-3366 of https://svn02/vc/mex_svn_test/S
Merging r3362                       H:\Luis_Martinez\trunk_dev_A
Updated                             H:\Luis_Martinez\trunk_dev_A\Bar.c
Recording mergeinfo for merge of r3362   H:\Luis_Martinez\trunk_dev_A
Updated                             H:\Luis_Martinez\trunk_dev_A
Merging r3364                       H:\Luis_Martinez\trunk_dev_A
Merged                              H:\Luis_Martinez\trunk_dev_A\Bar.c
Recording mergeinfo for merge of r3364   H:\Luis_Martinez\trunk_dev_A
Merged                              H:\Luis_Martinez\trunk_dev_A
Merging r3366                       H:\Luis_Martinez\trunk_dev_A
Merged                              H:\Luis_Martinez\trunk_dev_A\Bar.c
Recording mergeinfo for merge of r3366   H:\Luis_Martinez\trunk_dev_A
Merged                              H:\Luis_Martinez\trunk_dev_A
                                    H:\Luis_Martinez\trunk_dev_A

Completed!
```

Merged:4 Updated:2

## Reintegrate branch to trunk

When new features are finished in a branch, usually those changes will be reintegrated to trunk. In this case you should have the branch as working copy, and not the trunk.

1. The first step is be sure that your changes are committed, then use **TortoiseSVN** → **Switch** command from context menu.
2. Then, in the field **To path** set the trunk's URL, and configure the desired configuration, next, click **OK**.


H:\3D_MEX_TRAININ_SVN\trunk - Switch To Branch / Tag - TortoiseSVN

Switch:
H:/3D_MEX_TRAININ_SVN/trunk

To path: /cars

Source URL:
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk

Destination URL:
https://svn02/vc/mex_svn_test/cars

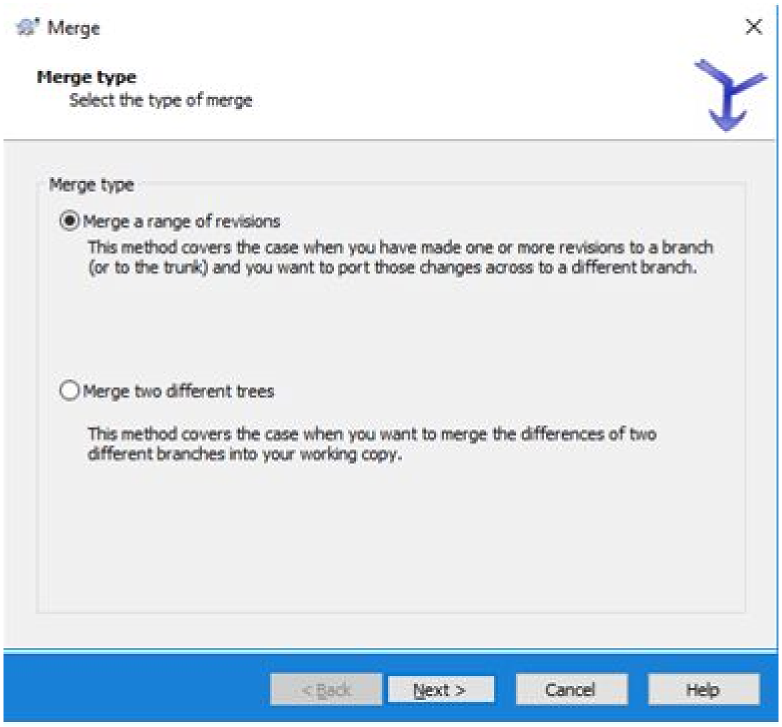
Revision: [✓ HEAD revision] Revision

Switch Depth: Working copy

☐ Make depth sticky
☐ Omit externals
☐ Ignore ancestry

After switch, your working copy will be on trunk.

3. Next, right click on your working copy and go to **TortoiseSVN** → **Merge** on context menu.
4. Select on new window **Merge range of revisions**, and click on **Next** button.


Merge

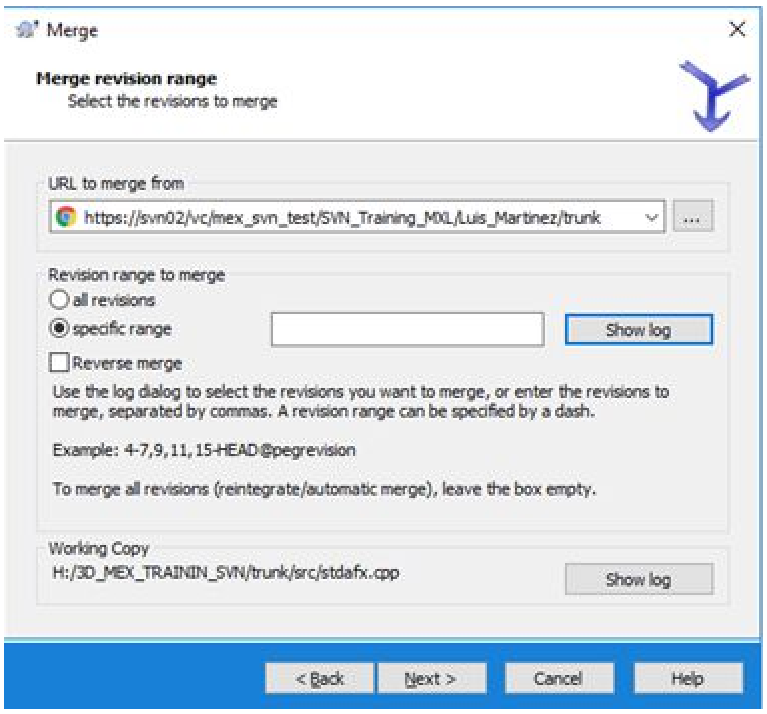
**Merge type**
Select the type of merge

Merge type

[✓ Merge a range of revisions]
This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

Merge two different trees
This method covers the case when you want to merge the differences of two different branches into your working copy.

5. On the next window add the full folder URL of the branch containing the changes you want to merge into your working copy ( trunk ). You may also click ... to browse the repository and find the desired branch.  Then click on **Next** button.


Merge

**Merge revision range**
Select the revisions to merge

URL to merge from
https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Martinez/trunk

Revision range to merge
all revisions
[✓ specific range]
Reverse merge

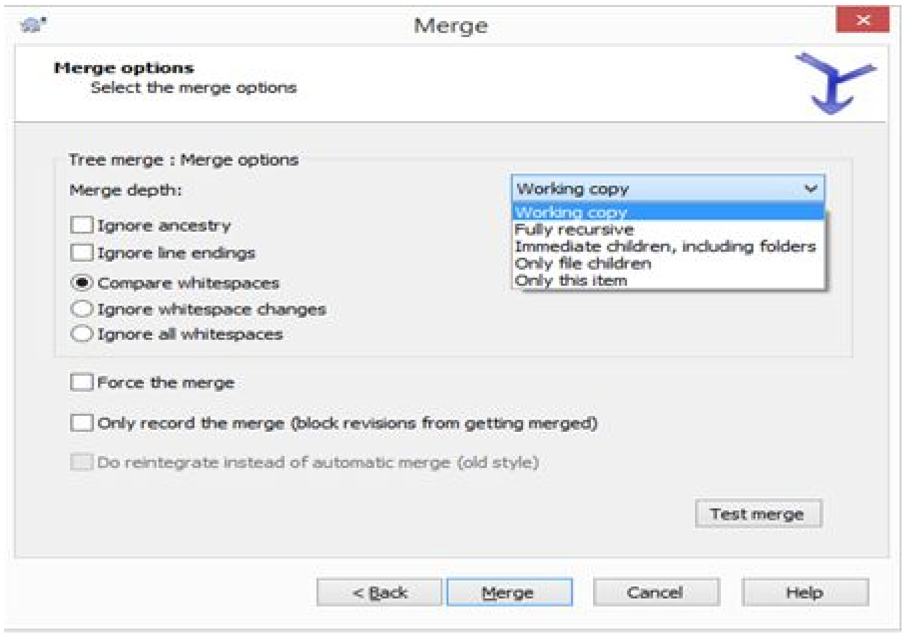
Use the log dialog to select the revisions you want to merge, or enter the revisions to merge, separated by commas. A revision range can be specified by a dash.

Example: 4-7,9,11,15-HEAD@pegrevision

To merge all revisions (reintegrate/automatic merge), leave the box empty.

Working Copy
H:/3D_MEX_TRAININ_SVN/trunk/src/stdafx.cpp

6. Next, set merge options in new window, then click on **Merge** button.


Merge

**Merge options**
Select the merge options

Tree merge : Merge options
Merge depth: Working copy

Working copy
Fully recursive
Immediate children, including folders
Only file children
Only this item

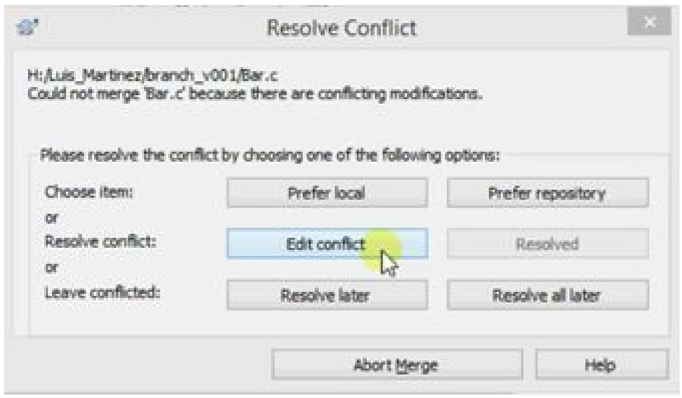
Ignore ancestry
Ignore line endings
[✓ Compare whitespaces]
Ignore whitespace changes
Ignore all whitespaces

Force the merge

Only record the merge (block revisions from getting merged)

Do reintegrate instead of automatic merge (old style)

7. Merge should be doing fine, but sometimes you will get a conflict (because you want to merge on the same code lines/parts in file where other developer  made changes too) .


Resolve Conflict

H:/Luis_Martinez/branch_v001/Bar.c
Could not merge 'Bar.c' because there are conflicting modifications.

Please resolve the conflict by choosing one of the following options:

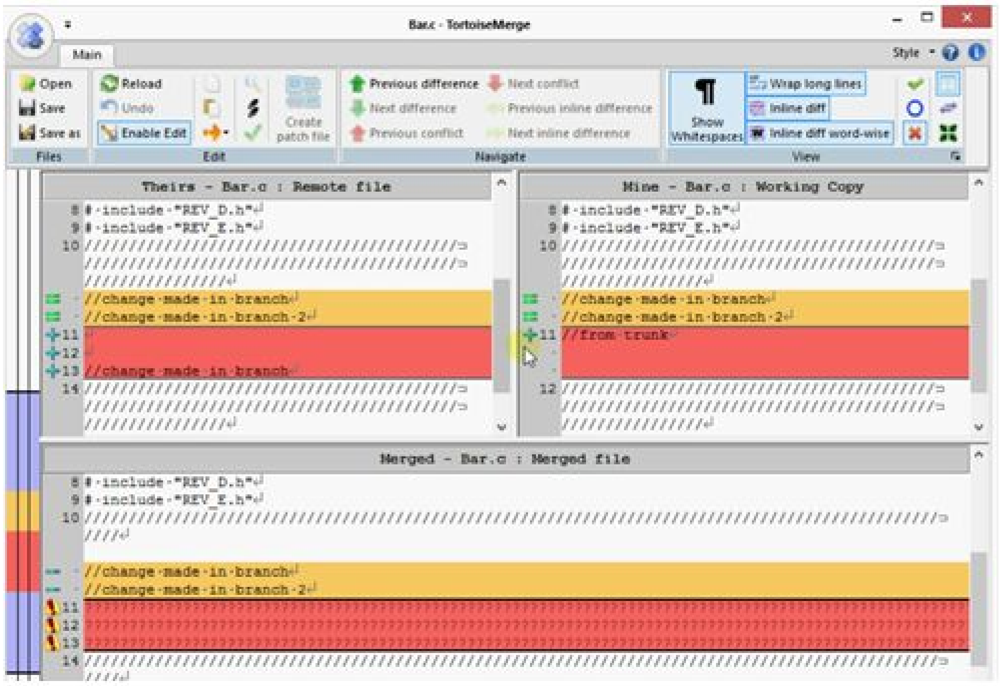
Choose item: Prefer local | Prefer repository
or
Resolve conflict: Edit conflict | Resolved
or
Leave conflicted: Resolve later | Resolve all later

Merge Conflict Callback Dialog

a. if you are merging text files then these first two buttons allow you to merge non-conflicting lines as normal and always prefer one version where there are conflicts. Choosing **Prefer local** will select your local version in every conflict, i.e. it will prefer what was already there before the merge over the incoming change from the merge source. Likewise, **Prefer repository** will select the repository changes in every conflict, i.e. it will prefer the incoming changes from the merge source over what was already in your working copy. This sounds easy, but the conflicts often cover more lines than you think they will and you may get unexpected results.

If your merge includes binary files, merging of conflicts in those is not possible in a line-by-line mode. A conflict in a binary file always refers to the complete file. Use **Prefer local** to select the local version as it was in your working copy prior to the merge, or **Prefer repository** to select the incoming file from the merge source in the repository.

b. Normally you will want to look at the conflicts and resolve them yourself. In that case, choose the **Edit Conflict** which will start up your merge tool. When you are satisfied with the result, click **Resolved**.



Theirs - Bar.c : Remote file

```
#include "REV_D.h"
#include "REV_E.h"
////////////////////////////////////////////
////////////////////////////////////////////
////////////////
//change made in branch
//change made in branch 2


//change made in branch
////////////////////////////////////////////
////////////////////////////////////////////
////////////////
```

Mine - Bar.c : Working Copy

```
#include "REV_D.h"
#include "REV_E.h"
////////////////////////////////////////////
////////////////////////////////////////////
////////////////
//change made in branch
//change made in branch 2
//from trunk
////////////////////////////////////////////
////////////////////////////////////////////
////////////////
```

Merged - Bar.c : Merged file

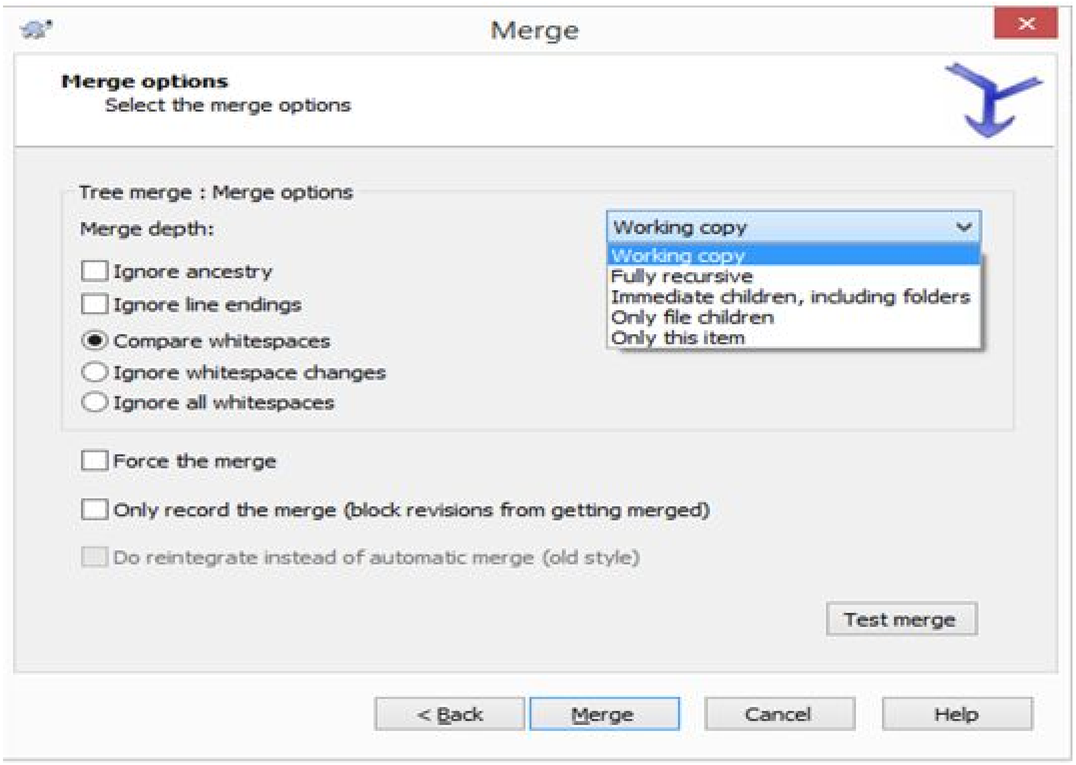
```
#include "REV_D.h"
#include "REV_E.h"
//////////////////////////////////////////////////////////////////////////////////////////
////
//change made in branch
//change made in branch 2
????????????????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????????????????
//////////////////////////////////////////////////////////////////////////////////////////
////
```

c.The last option is to postpone resolution and continue with merging. You can choose to do that for the current conflicted file, or for all files in the rest of the merge. However, if there are further changes in that file, it will not be possible to complete the merge.

7. After this, your working copy is have been merged with the branch, and now you must check everything is ok.
8. Commit your working copy changes to SVN.
9. Delete the branch if you will not use it anymore.

## Merge options

This page of the wizard lets you specify advanced options, before starting the merge process. Most of the time you can just use the default settings.



The **Merge depth** terms used are described in checkout depth. The default depth is Working copy, which uses the existing depth setting.

**Ignore ancestry** box, is to make Subversion use only path-based differences rather than history-based differences.

**Ignore line endings,** excludes changes which are due solely to difference in line-end style.

**Compare whitespaces,** includes all changes in indentation and inline whitespace as added/removed lines.

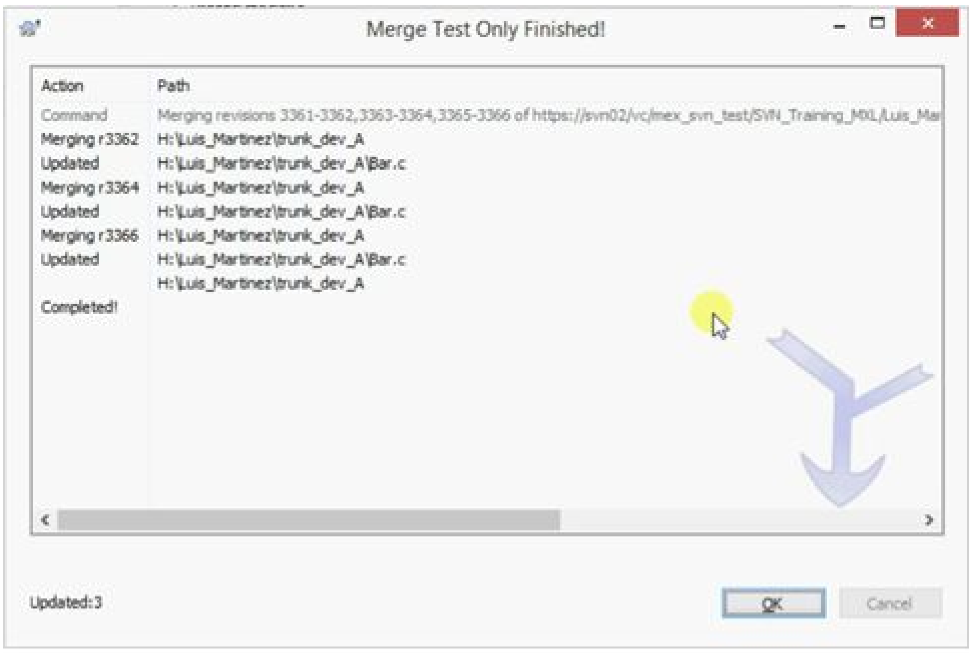
**Ignore whitespace changes,** excludes changes which are due solely to a change in the amount or type of white space, e.g. changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change.

**Ignore all whitespaces,** excludes all whitespace-only changes.

**Force the merge,** is used to avoid a tree conflict where an incoming delete affects a file that is either modified locally or not versioned at all. If the file is deleted then there is no way to recover it, which is why that option is not checked by default.

**Only record the merge** checkbox, use it if you are using merge tracking and you want to mark a revision as having been merged, without actually doing the merge here.

**Test Merge**, simulates the merge operation, but does not modify the working copy at all. It shows you a list of the files that will be changed by a real merge, and notes files where conflicts may occur.



| Action | Path |
|---|---|
| Command | Merging revisions 3361-3362,3363-3364,3365-3366 of https://svn02/vc/mex_svn_test/SVN_Training_MXL/Luis_Ma |
| Merging r3362 | H:\Luis_Martinez\trunk_dev_A |
| Updated | H:\Luis_Martinez\trunk_dev_A\Bar.c |
| Merging r3364 | H:\Luis_Martinez\trunk_dev_A |
| Updated | H:\Luis_Martinez\trunk_dev_A\Bar.c |
| Merging r3366 | H:\Luis_Martinez\trunk_dev_A |
| Updated | H:\Luis_Martinez\trunk_dev_A\Bar.c |
|  | H:\Luis_Martinez\trunk_dev_A |
| Completed! |  |

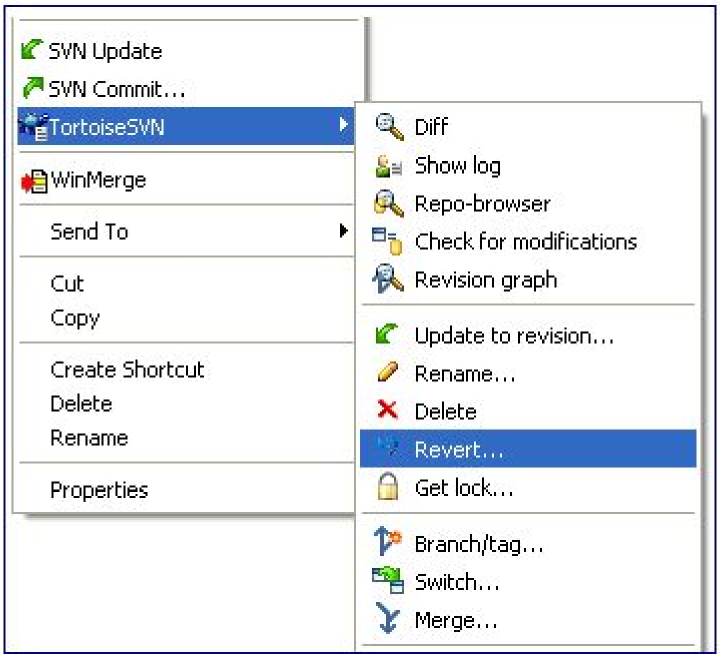
Updated:3

# How to revert

**Revert:** undo the changes you made to a file or a set of files.
There are 2 types of revert: revert to the base revision and revert to any revision.

## To the BASE revision.

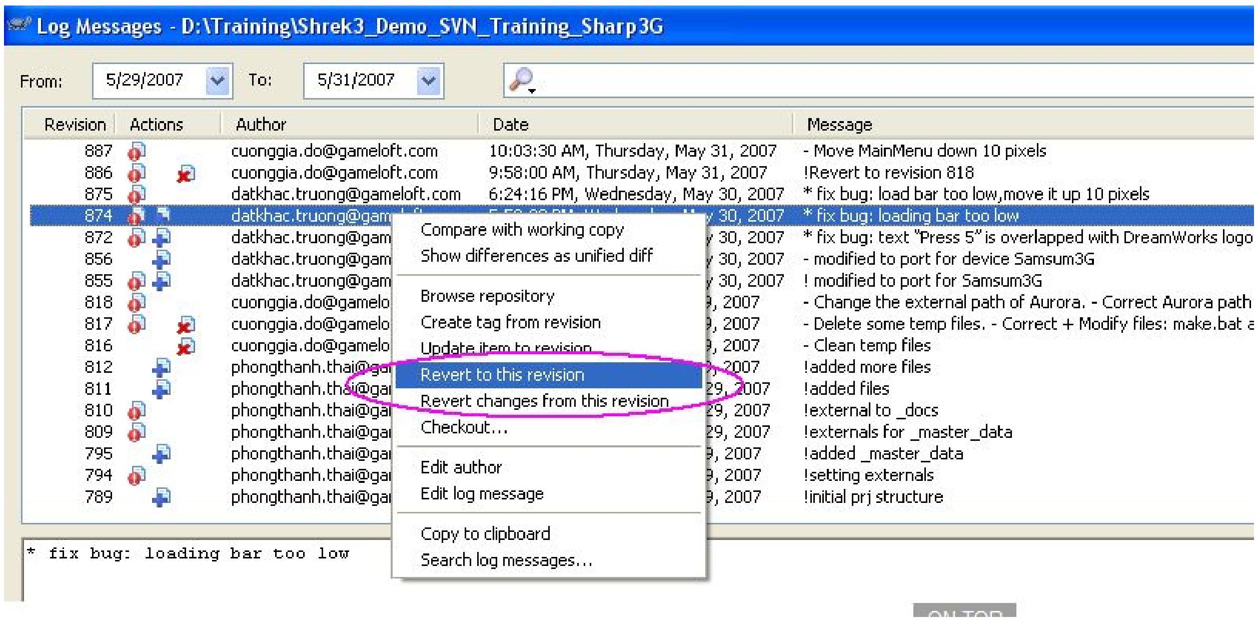
Subversion keeps a local "pristine" copy of each file as it was when you last updated your working copy. If you have made changes and decide you want to undo them, you can use the "revert" command to go back to the pristine copy.



## To any previous revision.

### Revert to this revision:

Revert to a specific revision

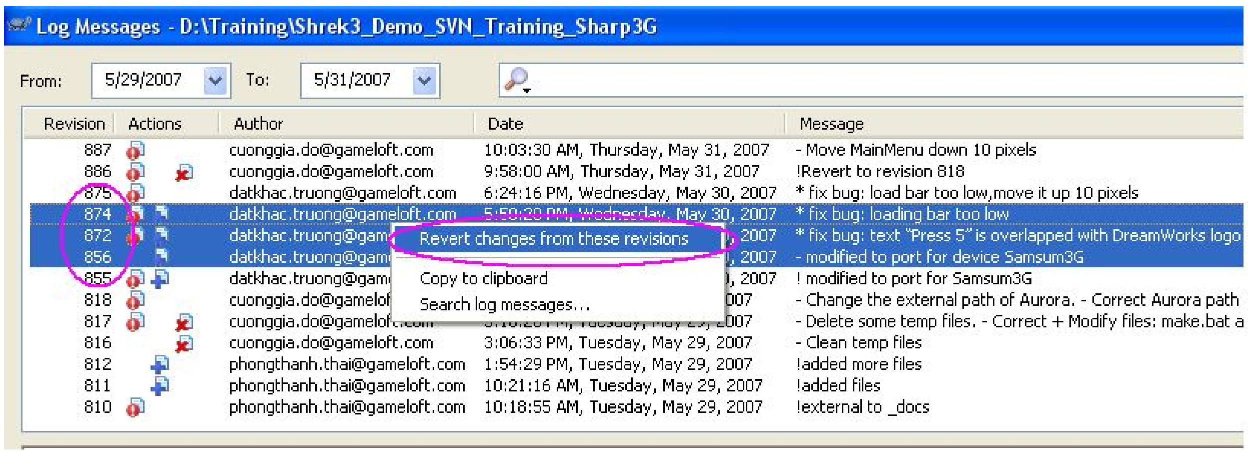


| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 887 | | cuonggia.do@gameloft.com | 10:03:30 AM, Thursday, May 31, 2007 | - Move MainMenu down 10 pixels |
| 886 | | cuonggia.do@gameloft.com | 9:58:00 AM, Thursday, May 31, 2007 | !Revert to revision 818 |
| 875 | | datkhac.truong@gameloft.com | 6:24:16 PM, Wednesday, May 30, 2007 | * fix bug: load bar too low,move it up 10 pixels |
| 874 | | datkhac.truong@gam | y 30, 2007 | * fix bug: loading bar too low |
| 872 | | datkhac.truong@gam | y 30, 2007 | * fix bug: text "Press 5" is overlapped with DreamWorks logo |
| 856 | | datkhac.truong@gam | y 30, 2007 | - modified to port for device Samsum3G |
| 855 | | datkhac.truong@gam | y 30, 2007 | ! modified to port for Samsum3G |
| 818 | | cuonggia.do@gamelo | 9, 2007 | - Change the external path of Aurora. - Correct Aurora path |
| 817 | | cuonggia.do@gamelo | 9, 2007 | - Delete some temp files. - Correct + Modify files: make.bat a |
| 816 | | cuonggia.do@gamelo | 9, 2007 | - Clean temp files |
| 812 | | phongthanh.thai@gai | 2007 | !added more files |
| 811 | | phongthanh.thai@gai | 29, 2007 | !added files |
| 810 | | phongthanh.thai@gai | 29, 2007 | !external to _docs |
| 809 | | phongthanh.thai@gai | 29, 2007 | !externals for _master_data |
| 795 | | phongthanh.thai@gai | 9, 2007 | !added _master_data |
| 794 | | phongthanh.thai@gai | 9, 2007 | !setting externals |
| 789 | | phongthanh.thai@gai | 9, 2007 | !initial prj structure |

```
* fix bug: loading bar too low
```

### Revert changes from these revisions

undo all actions in some revisions



Log Messages - D:\Training\Shrek3_Demo_SVN_Training_Sharp3G
From: 5/29/2007
To: 5/31/2007
Revert changes from these revisions
Copy to clipboard
Search log messages...

| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 887 | | cuonggia.do@gameloft.com | 10:03:30 AM, Thursday, May 31, 2007 | - Move MainMenu down 10 pixels |
| 886 | | cuonggia.do@gameloft.com | 9:58:00 AM, Thursday, May 31, 2007 | !Revert to revision 818 |
| 875 | | datkhac.truong@gameloft.com | 6:24:16 PM, Wednesday, May 30, 2007 | * fix bug: load bar too low,move it up 10 pixels |
| 874 | | datkhac.truong@gameloft.com | 5:50:20 PM, Wednesday, May 30, 2007 | * fix bug: loading bar too low |
| 872 | | datkhac.truong@gam | , 2007 | * fix bug: text "Press 5" is overlapped with DreamWorks logo |
| 856 | | datkhac.truong@gam | , 2007 | - modified to port for device Samsum3G |
| 855 | | datkhac.truong@gam | , 2007 | ! modified to port for Samsum3G |
| 818 | | cuonggia.do@gamelof | 007 | - Change the external path of Aurora. - Correct Aurora path |
| 817 | | cuonggia.do@gamelof | , 2007 | - Delete some temp files. - Correct + Modify files: make.bat a |
| 816 | | cuonggia.do@gameloft.com | 3:06:33 PM, Tuesday, May 29, 2007 | - Clean temp files |
| 812 | | phongthanh.thai@gameloft.com | 1:54:29 PM, Tuesday, May 29, 2007 | !added more files |
| 811 | | phongthanh.thai@gameloft.com | 10:21:16 AM, Tuesday, May 29, 2007 | !added files |
| 810 | | phongthanh.thai@gameloft.com | 10:18:55 AM, Tuesday, May 29, 2007 | !external to _docs |