TortoiseSVN
A Subversion client for Windows

Stefan Küng
Lübbe Onken
Simon Large
# Table of Contents

Preface .................................................................................................................. ix  
1. Audience ........................................................................................................... ix  
2. TortoiseSVN is free! ........................................................................................... ix  
3. Community ......................................................................................................... ix  
4. Acknowledgments .............................................................................................. ix  
5. Terminology used in this document ................................................................. x  

1. Introduction ...................................................................................................... 1  
  1.1. What is TortoiseSVN? ............................................................................. 1  
  1.2. TortoiseSVN's History ......................................................................... 1  
  1.3. TortoiseSVN's Features ....................................................................... 1  
  1.4. Installing TortoiseSVN ................................................................. 2  
    1.4.1. System requirements .............................................................. 2  
    1.4.2. Installation ............................................................................... 2  

2. Basic Concepts .................................................................................................. 4  
  2.1. The Repository ....................................................................................... 4  
  2.2. Versioning Models ............................................................................... 4  
    2.2.1. The Problem of File-Sharing .................................................... 5  
    2.2.2. The Lock-Modify-UnLock Solution ......................................... 5  
    2.2.3. The Copy-Modify-Merge Solution .......................................... 7  
    2.2.4. What does Subversion Do? ..................................................... 9  
  2.3. Subversion in Action .............................................................................. 9  
    2.3.1. Working Copies ....................................................................... 9  
    2.3.2. Revisions .................................................................................. 11  
    2.3.3. How Working Copies Track the Repository ............................ 13  
  2.4. Summary ............................................................................................... 13  

3. Setting Up A Server .......................................................................................... 14  
  3.1. Apache Based Server ........................................................................... 14  
    3.1.1. Introduction ............................................................................. 14  
    3.1.2. Installing Apache ..................................................................... 14  
    3.1.3. Installing Subversion ............................................................... 15  
    3.1.4. Configuration ........................................................................... 16  
    3.1.5. Authentication With A Windows Domain ............................... 19  
    3.1.6. Securing the server with SSL .................................................. 21  
  3.2. Svnserve Based Server .......................................................................... 23  
    3.2.1. Introduction ............................................................................. 23  
    3.2.2. Installing svnserve ................................................................. 23  
    3.2.3. Running svnserve ..................................................................... 23  
    3.2.4. Authentication with svnserve ................................................ 24  
    3.2.5. Authentication with svn+ssh .................................................. 25  

4. The Repository ................................................................................................. 26  
  4.1. Repository Creation .............................................................................. 26  
    4.1.1. Creating a Repository with the Command Line Client .......... 26  
    4.1.2. Creating The Repository With TortoiseSVN .......................... 26  
    4.1.3. Local Access to the Repository ............................................. 27  
  4.2. Repository Backup .................................................................................. 27  
  4.3. Hook Scripts ........................................................................................... 28  
  4.4. Checkout Links ....................................................................................... 28  

5. Daily Use Guide ............................................................................................... 29  
  5.1. Getting Started ....................................................................................... 29  
    5.1.1. Context Menus ........................................................................ 29  
    5.1.2. Drag and Drop ......................................................................... 30  
    5.1.3. Authentication ......................................................................... 31  
  5.2. Importing Data Into A Repository ...................................................... 32  
    5.2.1. Repository Layout ................................................................. 32  
    5.2.2. Import .................................................................................... 33  
    5.2.3. Special Files .......................................................................... 34  
    5.2.4. Referenced Projects .............................................................. 35
B.4.1. Use the revision log dialog ............................................................89
B.4.2. Use the merge dialog .................................................................90
B.4.3. Use svndumpfilter .................................................................90
B.5. Compare two revisions of a file .....................................................90
B.6. Include a common sub-project ...................................................91
  B.6.1. Use svn:externals .................................................................91
  B.6.2. Use a nested working copy ...................................................91
  B.6.3. Use a relative location ..........................................................91
B.7. Create a shortcut to a repository ..................................................92
C. Automating TortoiseSVN .................................................................93
  C.1. TortoiseSVN Commands ........................................................93
Index ........................................................................................................96
List of Figures

2.1. A Typical Client/Server System ................................................................. 4
2.2. The Problem to Avoid ............................................................................. 5
2.3. The Lock-Modify-Unlock Solution .......................................................... 6
2.4. The Copy-Modify-Merge Solution ............................................................. 7
2.5. ...Copy-Modify-Merge Continued ............................................................ 8
2.6. The Repository's Filesystem .................................................................... 10
2.7. The Repository ....................................................................................... 12
4.1. The TortoiseSVN menu for unversioned folders ....................................... 26
5.1. Context menu for a directory under version control ................................ 30
5.2. Right drag menu for a directory under version control ............................ 31
5.3. Authentication Dialog ............................................................................. 31
5.4. The Import dialog .................................................................................. 34
5.5. The Checkout dialog ............................................................................... 35
5.6. The Commit dialog .................................................................................. 37
5.7. The Progress dialog showing a commit in progress .................................. 38
5.8. Progress dialog showing finished update ................................................. 39
5.9. Explorer showing icon overlays ............................................................... 42
5.10. Check for Modifications ........................................................................ 44
5.11. The Revision Log Dialog Top Pane with Context Menu ...................... 46
5.12. The Revision Log Dialog Top Pane with Context Menu ...................... 47
5.13. Top Pane Context Menu for 2 Selected Files ........................................ 48
5.14. The Log Dialog Bottom Pane with Context Menu .............................. 48
5.15. Explorer context menu for unversioned files ........................................ 52
5.16. Explorer context menu for unversioned files ........................................ 52
5.17. Explorer context menu for versioned files ............................................ 54
5.18. Revert dialog ....................................................................................... 55
5.19. Subversion property page .................................................................... 57
5.20. The Branch/Tag Dialog ....................................................................... 60
5.21. The Switch Dialog ............................................................................... 62
5.22. The Merge Dialog ................................................................................ 63
5.23. The Locking Dialog ............................................................................. 67
5.24. The Check for Modifications Dialog ..................................................... 68
5.25. The Annotate / Blame Dialog ............................................................... 70
5.26. TortoiseBlame .................................................................................... 71
5.27. The Repository Browser ....................................................................... 72
5.28. A Revision Graph ................................................................................ 73
5.29. The Relocate Dialog ............................................................................ 74
5.30. The Settings Dialog, General Page ....................................................... 77
5.31. The Settings Dialog, Look and Feel Page ............................................. 79
5.32. The Settings Dialog, Look and Feel Page ............................................. 80
5.33. The Settings Dialog, Icon Set Page ...................................................... 81
5.34. The Settings Dialog, Dialogs Page ........................................................ 81
5.35. The Settings Dialog, Network Page ...................................................... 83
5.36. The Settings Dialog, Diff Viewer Page .................................................. 84
5.37. The Settings Dialog, Diff/Merge Advanced Dialog ............................... 86
List of Tables

2.1. Repository Access URLs ................................................................. 10
3.1. Apache httpd.conf Settings ............................................................. 17
C.1. List of available command line options ........................................... 94
 Preface

• Do you work in a team?
• Has it ever happened that you were working on a file, and someone else was working on the same file at the same time? Did you lose your changes to that file because of that?
• Have you ever saved a file, and then wanted to revert the changes you made? Have you ever wished you could see what a file looked like some time ago?
• Have you ever found a bug in your project and wanted to know when that bug got into your files?

If you answered “yes” to one of these questions, then TortoiseSVN is for you! Just read on to find out how TortoiseSVN can help you in your work. It's not that difficult.

1. Audience

This book is written for computer literate folk who want to use Subversion to manage their data, but are uncomfortable using the command line client to do so. Since TortoiseSVN is a windows shell extension it's assumed that the user is familiar with the windows explorer and knows how to use it.

Even most Subversion users will never have to set up a server themselves. A whole chapter deals with how to set up such a server.

If you run into problems installing or working with TortoiseSVN, check Appendix A, Frequently Asked Questions (FAQ) at the end of this manual. If your question is not answered there, search our online FAQ at Berlios.de [http://tortoisesvn.berlios.de/faq/], which contains more information and is - due to its nature - more up to date.

2. TortoiseSVN is free!

TortoiseSVN is free. You don't have to pay for it, you can use it any way you want. It is developed under the GPL license (GPL).

TortoiseSVN is an Open Source project. That means you have full access to the source code of this program. You can browse it on this link http://svn.collab.net/repos/tortoisesvn/ [http://svn.collab.net/repos/tortoisesvn/]. The most recent version (where we're working on) is located under /trunk/, the released versions are located under /tags/.

3. Community

Both TortoiseSVN and Subversion are developed by a community of people who are working on those projects. They come from different countries all over the world and joined together to create wonderful programs.

4. Acknowledgments

Tim Kemp

for founding the TortoiseSVN project
Stefan Küng
for the hard work to get TortoiseSVN to what it is now

Lübbe Onken
for the beautiful icons, logo, bughunting and taking care of the documentation

The Subversion Book
for the great introduction to Subversion and its chapter 2 which we copied here

The Tigris Style project
for some of the styles which are reused in this documentation

Our Contributors
for the patches, bug reports and new ideas, and for helping others by answering questions on our mailing list.

Our Donators
for many hours of joy with the music they sent us

5. Terminology used in this document

To make reading the docs easier, the names of all the screens and Menus from TortoiseSVN are marked up in a different font. The Log Dialog for instance.

A menu choice is indicated with an arrow. TortoiseSVN->Show Log means: select Show Log from the TortoiseSVN context menu.

Where a local context menu appears within one of the TortoiseSVN dialogs, it is shown like this: Context Menu->Save As ...

User Interface Buttons are indicated like this: Press OK to continue.

User Actions are indicated using a bold font. ALT+A: press the ALT-Key on your keyboard and while holding it down press the A-Key as well. Right-drag: press the right mouse button and while holding it down drag the items to the new location.

System output and keyboard input is indicated with a different font as well.

Important

Important notes are marked with an icon.

Tip

Tips that make your life easier.

Caution

Places where you have to be careful what you are doing.

Warning

Where extreme care has to be taken, data corruption or other nasty things may occur if these warnings are ignored.
Chapter 1. Introduction

Version control is the art of managing changes to information. It has long been a critical tool for programmers, who typically spend their time making small changes to software and then undoing those changes the next day. Imagine a team of these programmers working concurrently - and perhaps even simultaneously at the very same files! - and you can see why a good system is needed to manage the potential chaos.

1.1. What is TortoiseSVN?

TortoiseSVN is a free open-source client for the Subversion version control system. That is, TortoiseSVN manages files and directories over time. Files are stored in a central repository. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your files and examine the history of how and when your data changed. This is why many people think of Subversion and version control systems in general as a sort of "time machine".

Some version control systems are also software configuration management (SCM) systems. These systems are specifically tailored to manage trees of source code, and have many features that are specific to software development - such as natively understanding programming languages, or supplying tools for building software. Subversion, however, is not one of these systems; it is a general system that can be used to manage any collection of files, including source code.

1.2. TortoiseSVN's History

In 2002, Tim Kemp found that Subversion was a very good version control system, but it lacked a good GUI client. The idea for a Subversion client as a Windows shell integration was inspired by the similar client for CVS named TortoiseCVS.

Tim studied the sourcecode of TortoiseCVS and used it as a base for TortoiseSVN. He then started the project, registered the domain tortoisesvn.org and put the sourcecode online. During that time, Stefan Küng was looking for a good and free version control system and found Subversion and the source for TortoiseSVN. Since TortoiseSVN was still not ready for use then he joined the project and started programming. Soon he rewrote most of the existing code and started adding commands and features, up to a point where nothing of the original code remained.

As Subversion became more stable it attracted more and more users who also started using TortoiseSVN as their Subversion client. The userbase grew quickly (and is still growing every day). That's when Lübbe Onken offered to help out with some nice icons and a logo for TortoiseSVN. And he takes care of the website and manages the translation.

1.3. TortoiseSVN's Features

What makes TortoiseSVN such a good Subversion client? Here's a short list of features.

- Shell integration
  TortoiseSVN integrates seamlessly into the Windows shell (i.e. the explorer). This means you can keep working with the tools you're already familiar with. And you do not have to change into a different application each time you need functions of the version control!

- And you are not even forced to use the Windows Explorer. TortoiseSVN's context menus work in many other file managers, and in the File/Open dialog which is common to most standard Windows applications. You should, however, bear in mind that TortoiseSVN is intentionally developed as extension for the Windows Explorer. Thus it is possible that in other applications the integration is not as complete and e.g. the icon overlays may not be shown.

- Icon overlays
  The status of every versioned file and folder is indicated by small overlay icons. That way you can see right away what the status of your working copy is.
Easy access to Subversion commands

All Subversion commands are available from the explorer context menu. TortoiseSVN adds its own submenu there.

Since TortoiseSVN is a Subversion client, we would also like to show you some of the features of Subversion itself:

Directory versioning

CVS only tracks the history of individual files, but Subversion implements a “virtual” versioned filesystem that tracks changes to whole directory trees over time. Files and directories are versioned. As a result, there are real client-side move and copy commands that operate on files and directories.

Atomic commits

A commit either goes into the repository completely, or not at all. This allows developers to construct and commit changes as logical chunks.

Versioned metadata

Each file and directory has an invisible set of “properties” attached. You can invent and store any arbitrary key/value pairs you wish. Properties are versioned over time, just like file contents.

Choice of network layers

Subversion has an abstracted notion of repository access, making it easy for people to implement new network mechanisms. Subversion’s “advanced” network server is a module for the Apache web server, which speaks a variant of HTTP called WebDAV/DeltaV. This gives Subversion a big advantage in stability and interoperability, and provides various key features for free: authentication, authorization, wire compression, and repository browsing, for example. A smaller, standalone Subversion server process is also available. This server speaks a custom protocol which can be easily tunneled over ssh.

Consistent data handling

Subversion expresses file differences using a binary differencing algorithm, which works identically on both text (human-readable) and binary (human-unreadable) files. Both types of files are stored equally compressed in the repository, and differences are transmitted in both directions across the network.

Efficient branching and tagging

The cost of branching and tagging need not be proportional to the project size. Subversion creates branches and tags by simply copying the project, using a mechanism similar to a hard-link. Thus these operations take only a very small, constant amount of time, and very little space in the repository.

Hackability

Subversion has no historical baggage; it is implemented as a collection of shared C libraries with well-defined APIs. This makes Subversion extremely maintainable and usable by other applications and languages.

1.4. Installing TortoiseSVN

1.4.1. System requirements

TortoiseSVN runs on Win2k SP2, WinXP or higher.

1.4.2. Installation

TortoiseSVN comes with an easy to use installer. Doubleclick on the installer file and follow the instructions. The installer will take care of the rest. Make sure to select the Install for all users option, unless you really want to restrict its use to the current user only.
If you want to use a spell checker for your log messages then you have to download the required dictionaries yourself. You can use the same dictionary files as OpenOffice [http://openoffice.org] and Mozilla [http://mozilla.org] use. You can find the required dictionary files in various places:

- http://lingucomponent.openoffice.org/spell_dic.html
  [http://lingucomponent.openoffice.org/spell_dic.html]

You can also use the dictionaries which get installed with Mozilla. They’re located in the installation folder of Mozilla.

Once you have got the dictionary files, you probably need to rename them so that the filenames only have the locale chars in it. Example:

- en_US.aff
- en_US.dic

Then just copy them to the bin sub-folder of the TortoiseSVN installation folder. Normally this will be C:\Program Files\TortoiseSVN\bin. If you don't want to litter the bin sub-folder, you can instead place your spell checker files in C:\Program Files\TortoiseSVN\Languages. If that folder isn't there, you have to create it first. The next time you start TortoiseSVN, the spell checker will be available.

If you encounter any problems during or after installing TortoiseSVN please refer to Appendix A, Frequently Asked Questions (FAQ) first.
Chapter 2. Basic Concepts

This chapter is a slightly modified version of the same chapter in the Subversion book. You can read the Subversion book here: http://svnbook.red-bean.com/ [http://svnbook.red-bean.com/].

This chapter is a short, casual introduction to Subversion. If you're new to version control, this chapter is definitely for you. We begin with a discussion of general version control concepts, work our way into the specific ideas behind Subversion, and show some simple examples of Subversion in use.

Even though the examples in this chapter show people sharing collections of program source code, keep in mind that Subversion can manage any sort of file collection - it's not limited to helping computer programmers.

2.1. The Repository

Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a filesystem tree - a typical hierarchy of files and directories. Any number of clients connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others.

Figure 2.1. A Typical Client/Server System

So why is this interesting? So far, this sounds like the definition of a typical file server. And indeed, the repository is a kind of file server, but it's not your usual breed. What makes the Subversion repository special is that it remembers every change ever written to it: every change to every file, and even changes to the directory tree itself, such as the addition, deletion, and rearrangement of files and directories.

When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But the client also has the ability to view previous states of the filesystem. For example, a client can ask historical questions like, "what did this directory contain last Wednesday?", or "who was the last person to change this file, and what changes did they make?" These are the sorts of questions that are at the heart of any version control system: systems that are designed to record and track changes to data over time.

2.2. Versioning Models

All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet? It's all too easy for users to accidentally overwrite each other's changes in the repository.
2.2.1. The Problem of File-Sharing

Consider this scenario: suppose we have two co-workers, Harry and Sally. They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, then it's possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file. While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made won't be present in Sally's newer version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively lost - or at least missing from the latest version of the file - and probably by accident. This is definitely a situation we want to avoid!

2.2.2. The Lock-Modify-Unlock Solution

Many version control systems use a lock-modify-unlock model to address this problem, which is a very simple solution. In such a system, the repository allows only one person to change a file at a time. First Harry must "lock" the file before he can begin making changes to it. Locking a file is a lot like borrowing a book from the library; if Harry has locked a file, then Sally cannot make any changes to it. If she tries to lock the file, the repository will deny the request. All she can do is read the file, and wait for Harry to finish his changes and release his lock. After Harry unlocks the file, his turn is over, and now Sally can take her turn by locking and editing.
The problem with the lock-modify-unlock model is that it’s a bit restrictive, and often becomes a roadblock for users:

- **Locking may cause administrative problems.** Sometimes Harry will lock a file and then forget about it. Meanwhile, because Sally is still waiting to edit the file, her hands are tied. And then Harry goes on vacation. Now Sally has to get an administrator to release Harry’s lock. The situation ends up causing a lot of unnecessary delay and wasted time.

- **Locking may cause unnecessary serialization.** What if Harry is editing the beginning of a text file, and Sally simply wants to edit the end of the same file? These changes don’t overlap at all. They could easily edit the file simultaneously, and no great harm would come, assuming the changes were properly merged together. There’s no need for them to take turns in this situation.

- **Locking may create a false sense of security.** Pretend that Harry locks and edits file A, while Sally simultaneously locks and edits file B. But suppose that A and B depend on one another, and the changes made to each are semantically incompatible. Suddenly A and B don’t work together anymore. The locking system was powerless to prevent the problem - yet it somehow provided a sense of false security. It’s easy for Harry and Sally to imagine that by locking files, each is beginning a safe, insulated task, and thus inhibits them from discussing their incompatible changes early on.
2.2.3. The Copy-Modify-Merge Solution

Subversion, CVS, and other version control systems use a *copy-modify-merge* model as an alternative to locking. In this model, each user's client reads the repository and creates a personal *working copy* of the file or project. Users then work in parallel, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly.

Here's an example. Say that Harry and Sally each create working copies of the same project, copied from the repository. They work concurrently, and make changes to the same file "A" within their copies. Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is *out-of-date*. In other words, that file A in the repository has somehow changed since he last copied it. So Harry asks his client to *merge* any new changes from the repository into his working copy of file A. Chances are that Sally's changes don't overlap with his own; so once he has both sets of changes integrated, he saves his working copy back to the repository.
But what if Sally’s changes do overlap with Harry’s changes? What then? This situation is called a conflict, and it’s usually not much of a problem. When Harry asks his client to merge the latest repository changes into his working copy, his copy of file A is somehow flagged as being in a state of conflict: he’ll be able to see both sets of conflicting changes, and manually choose between them. Note that software can’t automatically resolve conflicts; only humans are capable of understanding and making the necessary intelligent choices. Once Harry has manually resolved the overlapping changes (perhaps by discussing the conflict with Sally!), he can safely save the merged file back to the repository.

The copy-modify-merge model may sound a bit chaotic, but in practice, it runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don’t overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

In the end, it all comes down to one critical factor: user communication. When users communicate poorly, both syntactic and semantic conflicts increase. No system can force users to communicate perfectly, and no system can detect semantic conflicts. So there's no point in being lulled into a false promise that a locking system will somehow prevent conflicts; in practice, locking seems to inhibit productivity more than anything else.

There is one common situation where the lock-modify-unlock model comes out better, and that is where you have un-mergeable files. For example if your repository contains some graphic images, and two people change the image at the same time, there is no way for those changes to be merged together. Either Harry or Sally will lose their changes.
2.2.4. What does Subversion Do?

Subversion uses the copy-modify-merge solution by default, and in many cases this is all you will ever need. However, as of Version 1.2, Subversion also supports file locking, so if you have unmergeable files, or if you are simply forced into a locking policy by management, Subversion will still provide the features you need.

2.3. Subversion in Action

2.3.1. Working Copies

You've already read about working copies; now we'll demonstrate how the Subversion client creates and uses them.

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, and if they're source code files, you can compile your program from them in the usual way. Your working copy is your own private work area: Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so.

After you've made some changes to the files in your working copy and verified that they work properly, Subversion provides you with commands to "publish" your changes to the other people working with you on your project (by writing to the repository). If other people publish their own changes, Subversion provides you with commands to merge those changes into your working directory (by reading from the repository).

A working copy also contains some extra files, created and maintained by Subversion, to help it carry out these commands. In particular, each directory in your working copy contains a subdirectory named .svn, also known as the working copy administrative directory. The files in each administrative directory help Subversion recognize which files contain unpublished changes, and which files are out-of-date with respect to others' work.

A typical Subversion repository often holds the files (or source code) for several projects; usually, each project is a subdirectory in the repository's filesystem tree. In this arrangement, a user's working copy will usually correspond to a particular subtree of the repository.

For example, suppose you have a repository that contains two software projects.
In other words, the repository's root directory has two subdirectories: paint and calc.

To get a working copy, you must check out some subtree of the repository. (The term "check out" may sound like it has something to do with locking or reserving resources, but it doesn't; it simply creates a private copy of the project for you).

### 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:

<table>
<thead>
<tr>
<th>Schema</th>
<th>Access Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>file:///</td>
<td>Direct repository access on local disk.</td>
</tr>
<tr>
<td>https://</td>
<td>Same as http://, but with SSL encryption.</td>
</tr>
<tr>
<td>svn://</td>
<td>Unauthenticated TCP/IP access via custom protocol to an svnserve server.</td>
</tr>
<tr>
<td>svn+ssh:///</td>
<td>authenticated, encrypted TCP/IP access via custom protocol to an svnserve server.</td>
</tr>
</tbody>
</table>

### Table 2.1. Repository Access URLs
For the most part, Subversion’s URLs use the standard syntax, allowing for server names and port numbers to be specified as part of the URL. Remember that the file: access method is valid only for locations on the same server as the client - in fact, in accordance with convention, the server name portion of the URL is required to be either absent or localhost. Take great care that the file: access method is not used to access BDB repositories located on drives shared over the network!

Also, users of the file: scheme on Windows platforms will need to use an unofficially “standard” syntax for accessing repositories that are on the same machine, but on a different drive than the client's current working drive. Either of the two following URL path syntaxes will work where X is the drive on which the repository resides:

    file:///X:/path/to/repos
    ...
    file:///X|/path/to/repos
    ...

Note that a URL uses ordinary slashes even though the native (non-URL) form of a path on Windows uses backslashes.

Suppose you make changes to button.c. Since the .svn directory remembers the file’s modification date and original contents, Subversion can tell that you've changed the file. However, Subversion does not make your changes public until you explicitly tell it to. The act of publishing your changes is more commonly known as committing (or checking in) changes to the repository.

To publish your changes to others, you can use Subversion's commit command.

Now your changes to button.c have been committed to the repository; if another user checks out a working copy of /calc, they will see your changes in the latest version of the file.

Suppose you have a collaborator, Sally, who checked out a working copy of /calc at the same time you did. When you commit your change to button.c, Sally's working copy is left unchanged; Subversion only modifies working copies at the user's request.

To bring her project up to date, Sally can ask Subversion to update her working copy, by using the Subversion update command. This will incorporate your changes into her working copy, as well as any others that have been committed since she checked it out.

Note that Sally didn't need to specify which files to update; Subversion uses the information in the .svn directory, and further information in the repository, to decide which files need to be brought up to date.

### 2.3.2. Revisions

An 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 commit’s changes take place, or none of them take place. Subversion tries to retain this atomicity in the face of program crashes, system crashes, network problems, and other users' actions.

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.

Figure 2.7. The Repository

Global Revision Numbers

Unlike those of many other version control systems, Subversion's 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!

It's important to note that working copies do not always correspond to any single revision in the repository; they may contain files from several different revisions. For example, suppose you check out a working copy from a repository whose most recent revision is 4:

```
calc/Makefile:4
  integer.c:4
  button.c:4
```

At the moment, this working directory corresponds exactly to revision 4 in the repository. However, suppose you make a change to button.c, and commit that change. Assuming no other commits have taken place, your commit will create revision 5 of the repository, and your working copy will now look like this:

```
calc/Makefile:4
  integer.c:4
  button.c:5
```

Suppose that, at this point, Sally commits a change to integer.c, creating revision 6. If you use
**svn update** to bring your working copy up to date, then it will look like this:

```
calc/Makefile:6
  integer.c:6
  button.c:6
```

Sally's changes to integer.c will appear in your working copy, and your change will still be present in button.c. In this example, the text of Makefile is identical in revisions 4, 5, and 6, but Subversion will mark your working copy of Makefile with revision 6 to indicate that it is still current. So, after you do a clean update at the top of your working copy, it will generally correspond to exactly one revision in the repository.

### 2.3.3. How Working Copies Track the Repository

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.

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 an **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. An **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. An **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.

### 2.4. Summary

We've covered a number of fundamental Subversion concepts in this chapter:

- We've introduced the notions of the central repository, the client working copy, and the array of repository revision trees.
- We've seen some simple examples of how two collaborators can use Subversion to publish and receive changes from one another, using the 'copy-modify-merge' model.
- We've talked a bit about the way Subversion tracks and manages information in a working copy.
Chapter 3. Setting Up A Server

To use TortoiseSVN (or any other Subversion client), you need a place where your repositories are located. You can either store your repositories locally and access them using the file:// protocol or you can place them on a server and access them with the http:// or svn:// protocols. The two server protocols can also be encrypted. You use https:// or svn+ssh://. This chapter shows you step by step on how you can set up such a server on a Windows machine.

If you don't have a server and/or if you only work alone then local repositories are probably your best choice. You can skip this chapter and go directly to Chapter 4, The Repository.

3.1. Apache Based Server

3.1.1. Introduction

The most flexible of all possible server setups for Subversion is the Apache based one. Although a bit more complicated to set up, it offers benefits that other servers cannot:

WebDAV
The Apache based Subversion server uses the WebDAV protocol which is supported by many other programs as well. You could e.g. mount such a repository as a "Webfolder" in the Windows explorer and then access it like any other folder in the filesystem

Browsing The Repository
You can point your browser to the URL of your repository and browse the contents of it without having a Subversion client installed. This gives access to your data to a much wider circle of users.

Authentication
You can use any authentication mechanism Apache supports, including SSPI and LDAP.

Security
Since Apache is very stable and secure, you automatically get the same security for your repository. This includes SSL encryption.

3.1.2. Installing Apache

The first thing you need before installing Apache is a computer with either Windows2000 / WinXP+SP1 or Windows2003.

Warning
Please note that Windows XP without the servicepack 1 will lead to bogus network data and could therefore corrupt your repository!

1. Download the latest version of the Apache webserver from http://httpd.apache.org/download.cgi. Make sure that you download the version > 2.0.54 - the version 1.3.xx won't work! Also, versions lower than 2.0.54 won't work with Subversion 1.2 because of a bug in how Apache < 2.0.54 was built for Windows.

2. Once you have the Apache2 installer you can doubleclick on it and it will guide you through the installation process. Make sure that you enter the server-URL correctly (if you don't have a dns name for your server just enter the ip-address). I recommend to install apache for All Users, on Port 80, as a Service. Note: if you already have IIS or any other program running which listens on port 80 the installation might fail. If that happens, go to the pro-
grams directory, \Apache Group\Apache2\conf and locate the file httpd.conf. Edit that file so that Listen 80 is changed to a free port, e.g. Listen 81. Then restart the installation - this time it should finish without problems.

3. Now test if the Apache-webserver is running correctly by pointing your webbrowser to http://localhost/- a preconfigured Website should show up.

Caution

If you decide to install Apache as a service, be warned that by default it will run as the local system account. It would be a more secure practice for you to create a separate account for Apache to run as.

Make sure that the account on the server that Apache is running as has an explicit entry in the repository directory's access control list (right-click directory | properties | security), with full control. Otherwise, users will not be able to commit their changes.

Even if Apache runs as local system, you still need such an entry (which will be the Administrator account in this case).

If Apache does not have this permission set up, your users will get "Access denied" error messages, which show up in the Apache error log as error 500.

3.1.3. Installing Subversion


2. Run the Subversion installer and follow the instructions. If the Subversion installer recognized that you’ve installed Apache, then you’re almost done. If it couldn’t find an Apache server then you have to do some additional steps.

3. Using the windows explorer, go to the installation directory of Subversion (usually c:\program files\Subversion) and find the files /httpd/mod_dav_svn.so and mod_authz_svn.so. Copy these files to the Apache modules directory (usually c:\program files\apache group\apache2\modules).

4. Copy the file /bin/libdb42.dll from the Subversion installation directory to the Apache modules directory.

5. Edit Apache's configuration file (usually C:\Program Files\Apache Group\Apache2\conf\httpd.conf) with a text editor such as Notepad and make the following changes:

   Uncomment (remove the '#' mark) the following lines:

   #LoadModule dav_fs_module modules/mod_dav_fs.so
   #LoadModule dav_module modules/mod_dav.so

   Add the following two lines to the end of the LoadModule section.

   LoadModule dav_svn_module modules/mod_dav_svn.so
   LoadModule authz_svn_module modules/mod_authz_svn.so
3.1.4. Configuration

Now you have set up Apache and Subversion, but Apache doesn't know how to handle Subversion clients like TortoiseSVN yet. To get Apache to know which URL shall be used for Subversion repositories you have to edit the Apache config file (usually located in `c:\program files\apache group\apache2\conf\httpd.conf`) with any text editor you like (e.g. Notepad):

1. At the end of the Config file add the following lines:

   ```
   <Location /svn>
   DAV svn
   SVNParentPath D:\SVN
   AuthType Basic
   AuthName "Subversion repositories"
   AuthUserFile passwd
   #AuthzSVNAccessFile svnaccessfile
   Require valid-user
   </Location>
   ```

   This configures Apache so that all your Subversion repositories are physically located below D:\SVN. The repositories are served to the outside world from the URL: http://MyServer/svn/. Access is restricted to known users/passwords listed in the passwd file.

2. To create the passwd file, open the command prompt (DOS-Box) again, change to the apache2 folder (usually `c:\program files\apache group\apache2`) and create the file by entering

   ```
   bin\htpasswd -c passwd <username>
   ```

   This will create a file with the name passwd which is used for authentication. Additional users can be added with

   ```
   bin\htpasswd passwd <username>
   ```

3. Restart the Apache service again.

4. Point your browser to http://MyServer/svn/MyNewRepository (where MyNewRepository is the name of the Subversion repository you created before). If all went well you should be prompted for a username and password, then you can see the contents of your repository.

A short explanation of what you just entered:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Location /svn&gt;</code></td>
<td>means that the Subversion repositories are available from the URL <a href="http://MyServer/svn/">http://MyServer/svn/</a></td>
</tr>
<tr>
<td>DAV svn</td>
<td>tells Apache which module will be responsible to serve that URL - in this case the Subversion module.</td>
</tr>
<tr>
<td>SVNParentPath D:\SVN</td>
<td>tells Subversion to look for repositories below D:\SVN</td>
</tr>
<tr>
<td>AuthType Basic</td>
<td>is to activate basic authentication, i.e. Username/password</td>
</tr>
<tr>
<td>AuthName &quot;Subversion repositories&quot;</td>
<td>is used as an information whenever an authentication dialog pops up to tell the user what the authentication is for</td>
</tr>
<tr>
<td>Setting</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AuthUserFile passwd</td>
<td>specifies which password file to use for authentication</td>
</tr>
<tr>
<td>AuthzSVNAccessFile</td>
<td>Location of the Access file for paths inside a Subversion repository</td>
</tr>
<tr>
<td>Require valid-user</td>
<td>specifies that only users who entered a correct username/password are allowed to access the URL.</td>
</tr>
</tbody>
</table>

Table 3.1. Apache httpd.conf Settings

But that’s just an example. There are many, many more possibilities of what you can do with the Apache webserver.

• If you want your repository to have read access for everyone but write access only for specific users you can change the line

  Require valid-user

to

  <LimitExcept GET PROPFIND OPTIONS REPORT>
  Require valid-user
  </LimitExcept>

• Using a passwd file limits and grants access to all of your repositories as a unit. If you want more control over which users have access each folder inside a repository you can uncomment the line

  #AuthzSVNAccessFile svnaccessfile

and create a Subversion access file. Apache will make sure that only valid users are able to access your /svn location, and will then pass the username to Subversion's AuthzSVNAccessFile module so that it can enforce more granular access based upon rules listed in the Subversion access file. An example file would look like this:

```
[groups]
developers = user1,user2,user3,user4
docs = user5,user6,user7
#to allow everyone read access
[/]
* = r
#allow all developers complete access
@developers = rw
#give the doc people write access to the docs folder
[/project/trunk/doc]
@docs = rw
```

Use SVNParentPath

If you used the SVNParentPath directive then you don't have to change the Apache config file everytime you add a new Subversion repository. Simply create the new repository under the same location as the first repository and you're done! In my company I have direct access to that specific folder on the server via SMB (normal windows file access). So I just create a new folder there, run the TortoiseSVN command
TortoiseSVN->Create repository here... and a new project has a home...

**Index of projects under SVNParentPath**

The advantage of using the SVNParentPath directive is that you don’t have to change
the Apache config each time you create a new repository, but you also don’t have an
index of all created and available projects. If you point your browser to the path SVN-
ParentPath points to, you will get a nasty error page showing.

To avoid that ugly error page and have a nice looking listing of all available projects
instead, you can use the following PHP script which generates the index for you auto-
matically. (You will need to install PHP on your server in order to use the below file).

```php
<html>
<head>
<title>Subversion Repositories</title>
</head>
<body>
<h2>Subversion Repositories</h2>
<p>
<?php
$svnparentpath = "C:/svn";
$svnparenturl = "/svn";

$dh = opendir( $svnparentpath );
if( $dh ) {
    while( $dir = readdir( $dh ) ) {
        $svndir = $svnparentpath . "/" . $dir;
        $svndbdir = $svndir . "/db";
        $svnfstypefile = $svndbdir . "/fs-type";
        if( is_dir( $svndir ) && is_dir( $svndbdir ) ) {
            echo "<a href=" . $svnparenturl . "/" . $dir . ">
            if( file_exists( $svnfstypefile ) ) {
                $handle = fopen ("$svnfstypefile", "r");
                $buffer = fgets($handle, 4096);
                fclose ( $handle );
                $buffer = chop ( $buffer );
                if( strcmp ( $buffer, "fsfs" )==0 ) {
                    echo " (FSFS) <br />
                } else {
                    echo " (BDB) <br />
                }
            } else {
                echo " (BDB) <br />
            }
        } else {
            echo " (BDB) <br />
        }
    }
    closedir( $dh );
}

Save the lines above to a file svn_index.php and store that file in your web root folder. Next
you have to tell Apache to show that page instead of the error:
3.1.5. Authentication With A Windows Domain

As you might have noticed you need to make a username/password entry in the passwd file for each user separately. And if (for security reasons) you want your users to periodically change their passwords you have to make the change manually.

But there's a solution for that problem - at least if you're accessing the repository from inside a LAN with a windows domain controller: mod_auth_sspi!

The original SSPI module was offered by Syneapps including sourcecode. But the development for it has been stopped. You can still find that module on the internet - just do a google search for it. We also have a compiled version of it on our website, including a patch from Vladimir Berezniker. Get the mod_auth_sspi module [http://tortoisesvn.tigris.org/mod_auth_sspi.zip].

• Download the module, copy the file mod_auth_sspi.so into the Apache modules folder.

• Edit the Apache config file: add the line

    LoadModule sspi_auth_module modules/mod_auth_sspi.so

to the LoadModule's section. Make sure you insert this line **before** the line

    LoadModule auth_module modules/mod_auth.so

• To make the Subversion location use this type of authentication you have to change the line

    AuthType Basic

to

    AuthType SSPI

also you need to add

    SSPIAuth On
    SSPIAuthoritative On
    SSPIDomain <domaincontroller>
    SSPIOfferBasic On

within the <Location /svn> block. If you don't have a domain controller, leave the name of the
domain control as <domaincontroller>.

Note that if you are authenticating using SSPI, then you don't need the AuthUserFile line to define a password file any more. Apache authenticates your username and password against your windows domain instead. You will need to update the users list in your svnaccessfile to reference DOMAIN\username as well.

Tip

Subversion AuthzSVNAccessFile files are case sensitive in regard to user names ("JUser" is different from "juser").

In Microsoft's world, Windows domains and usernames are not case sensitive. Even so, some network administrators like to create user accounts in CamelCase (e.g. "JUser").

This difference can bite you when using SSPI authentication as the windows domain and user names are passed to Subversion in the same case as the user types them in at the prompt. Internet Explorer often passes the username to Apache automatically using whatever case the account was created with.

The end result is that you may need at least two entries in your AuthzSVNAccessFile for each user -- a lowercase entry and an entry in the same case that Internet Explorer passes to Apache. You will also need to train your users to also type in their credentials using lower case when accessing repositories via TortoiseSVN.

Apache's Error and Access logs are your best friend in deciphering problems such as these as they will help you determine the username string passed onto Subversion's AuthzSVNAccessFile module. You may need to experiment with the exact format of the user string in the svnaccessfile (e.g. DOMAIN\user vs. DOMAIN//user) in order to get everything working.

SSL and InternetExplorer

If you're securing your server with SSL and use authentication against a windows domain you will encounter that browsing the repository with the Internet Explorer doesn't work anymore. Don't worry - this is only the Internet Explorer not able to authenticate. Other browsers don't have that problem and TortoiseSVN and any other Subversion client are still able to authenticate.

If you still want to use IE to browse the repository you can either:

- define a separate <Location /path> directive in the apache config file, and add the SSPIBasicPreferred On. This will allow IE to authenticate again, but other browsers and Subversion won't be able to authenticate against that location.
- Offer browsing with unencrypted authentication (without SSL) too. Strangely IE doesn't have any problems with authenticating if the connection is not secured with SSL.
- In the ssl "standard" setup there's often the following statement in apache's virtual ssl host:

  SetEnvIf User-Agent ".*MSIE.*"  
  nokeepalive ssl-unclean-shutdown \ 
  downgrade-1.0 force-response-1.0
There are (were?) good reasons for this configuration, see http://www.modssl.org/docs/2.8/ssl_faq.html#ToC49 [http://www.modssl.org/docs/2.8/ssl_faq.html#ToC49] But if you want ntlm authentication you have to use keepalive: http://www.microsoft.com/resources/documentation/WindowsServ/2003/standard/proddocs/en-us/qos_enablekeepalives.asp [http://www.microsoft.com/resources/documentation/WindowsServ/2003/standard/proddocs/en-us/qos_enablekeepalives.asp] If You uncomment the whole "SetEnvIf" You should be able to authenticate IE with windows authentication over SSL against the apache on Win32 with included mod_auth_sspi.

3.1.6. Securing the server with SSL

The apache server doesn't have SSL support installed by default due to US-export restrictions. But you can easily download the required module from somewhere else and install it yourself.

1. First you need the required files to enable SSL. You can find those in the package available at http://hunter.campbus.com/ [http://hunter.campbus.com/]. Just unzip the package and then copy mod_ssl.so to the modules folder of Apache and the file openssl.exe to the bin folder. Also copy the file conf/ssl.conf to the conf folder of Apache.

2. Open the file ssl.conf in the Apache conf folder with a text editor.

3. Place a comment char (#) in front of the following lines:

   DocumentRoot "c:/apache/htdocs"
   ServerName www.example.com:443
   ServerAdmin you@example.com
   ErrorLog logs/error_log
   TransferLog logs/access_log

4. change the line

   SSLCertificateFile conf/ssl.crt/server.crt

to

   SSLCertificateFile conf/ssl/my-server.cert

the line

   SSLCertificateKeyFile conf/ssl.key/server.key

to

   SSLCertificateKeyFile conf/ssl/my-server.key

and the line

   SSLMutex file:logs/ssl_mutex

to

   SSLMutex default
5. Delete the lines

\[\text{<IfDefine SSL>}
\\text{and}
\text{</IfDefine>}\]

6. Open the Apache config file (httpd.conf) and uncomment the line

\[
\text{#LoadModule ssl_module modules/mod_ssl.so}
\]

7. OpenSSL needs a config file. You can download a working one from http://tud.at/programm/openssl.cnf. Save the file to bin/openssl.cnf. Please note: the file has the type *.cnf. Windows treats such files in a special way but it really is just a text file!

8. Next you need to create an SSL certificate. To do that open a command prompt (DOS-Box) and change to the apache folder (e.g. C:\program files\apache group\apache2) and type the following command:

\[
\text{bin\openssl req -config bin\openssl.cnf -new -out my-server.csr}
\]

You will be asked for a passphrase. Please don't use simple words but whole sentences, e.g. a part of a poem. The longer the phrase the better. Also you have to enter the URL of your server. All other questions are optional but I recommend to fill out those too. Next type the commands

\[
\text{bin\openssl rsa -in privkey.pem -out my-server.key}
\]

and (on one line)

\[
\text{bin\openssl x509 -in my-server.csr -out my-server.cert -req -signkey my-server.key -days 4000}
\]

This will create a certificate which will expire in 4000 days. And finally enter:

\[
\text{bin\openssl x509 -in my-server.cert -out my-server.der.crt -outform DER}
\]

These commands created some files in the Apache folder (my-server.der.crt, my-server.csr, my-server.key, .rnd, privkey.pem, my-server.cert). Copy the files to the folder conf/ssl (e.g. C:\program files\apache group\apache2\conf\ssl)- if this folder does not exist you have to create it first.

9. Restart the apache service.

10. Point your browser to https://servername/svn/project ...

Forcing SSL access

When you've set up SSL to make your repository more secure, you might want to disable the normal access via non-ssl (http) and only allow https access. To do this, you
have to add another directive to the Subversion <Location> block: SSLRequireSSL.

An example <Location> block would look like this:

```xml
<Location /svn>
DAV svn
SVNPARENTPath D:\SVN
SSLRequireSSL
AuthType Basic
AuthName "Subversion repositories"
AuthUserFile passwd
#AuthzSVNAccessFile svnaccessfile
Require valid-user
</Location>
```

3.2. Svnserve Based Server

3.2.1. Introduction

There may be situations where it’s not possible to use Apache as your server. Fortunately, Subversion includes Svnserve - a lightweight stand-alone server which uses a custom protocol over an ordinary TCP/IP connection.

In most cases svnserve is easier to setup and runs faster than the Apache based server.

3.2.2. Installing svnserve

2. Run the Subversion installer. If you run the installer on your server you can skip step 3.
3. Open the windows-explorer, go to the installation directory of Subversion (usually C:\Program Files\Subversion) and in the bin directory, find the files svnserve.exe, libdb42.dll, libeay32.dll and ssleay32.dll - copy these files into a directory on your server e.g. c:\svnserve

3.2.3. Running svnserve

Now that svnserve is installed, you need it running on your server. The simplest approach is to run the following from a DOS shell or create a windows shortcut:

```
svnserve.exe -d
```

svnserve will now start waiting for incoming requests on port 3690. The -d switch tells svnserve to run as a daemon process, so it will always exist until it is manually terminated.

If you have not yet created a repository, follow the instructions given with the Apache server setup Section 3.1.4, “Configuration”.

To test that svnserve is working, use TortoiseSVN->Repo-Browser to view a repository.

Assuming your repository is located in c:\repos\TestRepo, and your server is called localhost, enter:
svn://localhost/repos/TestRepo
when prompted by the repo browser.

You can also increase security and save time entering URL's with svnservice by using the -r switch to
set the root location and restrict access to a specified directory on the server:

svnservice.exe -d -r drive:\path\to\repository

Using the previous test as a guide, svnservice would now run as:

svnservice.exe -d -r c:\repos

And in TortoiseSVN our repo-browser URL is now shortened to:

svn://localhost/TestRepo

Note that the -r switch is also needed if your repository is located on a different partition or drive
than the location of svnservice on your server.

---

**Warning**

Do not create or access a Berkeley DB repository on a network share. It cannot exist
on a remote filesystem. Not even if you have the network drive mapped to a drive let-
ter. If you attempt to use Berkeley DB on a network share, the results are unpredictable
- you may see mysterious errors right away, or it may be months before you discover
that your repository database is subtly corrupted.

---

**Run svnservice as a Service**

If you are concerned about always having a user logged in on your server, or worried
about someone shutting down svnservice or forgetting to restart it after a reboot, it is
possible to run svnservice as a windows service using a wrapper like the one available at
svnservice [http://dark.clansoft.dk/~mbn/svnservice/] provided by Magnus Norddahl.
Note that SVNService installs svnservice as a manual service, you may want to change
this to automatic.

More generic tools like firedaemon [http://www.firedaemon.com/] will also work.
Note that you will still need to run svnservice with the -d switch.

---

### 3.2.4. Authentication with svnservice

The default svnservice setup provides anonymous read-only access, so whilst you can use the reop-
browser in TortoiseSVN to view a repository using an svn:// URL, you won't be able to perform any
normal Subversion operations such as checkout, update or commit.

To enable write access to a repository, you need to edit the conf/svnservice.conf file in your
repository directory. This file controls the configuration of the svnservice daemon, and also contains
useful documentation.

You can enable anonymous write access by simply setting:

[general]
anon-access = write

However, you will not know who has made changes to a repository, as the svn:author property
will be empty. You will also be unable to control who makes changes to a repository. This is a somewhat risky setup!

One way to overcome this is to create a password database:

```plaintext
[general]
anon-access = none
auth-access = write
password-db = userfile
```

Where `userfile` is a file which exists in the same directory as `svnserve.conf`. This file can live elsewhere in your filesystem (useful for when you have multiple repositories which require the same access rights) and may be referenced using an absolute path, or a path relative to the `conf` directory. If you include a path, it must be written `/the/unix/way`. Using `\` or drive letters will not work. The `userfile` should have a structure of:

```plaintext
[users]
username = password
...
```

This example would deny all access for unauthenticated (anonymous) users, and give read-write access to users listed in `userfile`.

### Tip

If you maintain multiple repositories using the same password database, the use of an authentication realm will make life easier for users, as TortoiseSVN can cache your credentials so that you only have to enter them once. More information can be found in the Subversion book, specifically in the sections Create a 'users' file and realm [http://svnbook.red-bean.com/en/1.1/ch06s03.html#svn-ch-6-sect-3.2.1] and Client Credentials Caching [http://svnbook.red-bean.com/en/1.1/ch06s02.html#svn-ch-6-sect-2.2]

### 3.2.5. Authentication with svn+ssh

Another way to authenticate users with a svnserve based server is to use a secure shell (SSH) to tunnel requests through.

With this approach, svnserve is not run as a daemon process, rather, the secure shell starts svnserve for you, running it as the SSH authenticated user. To enable this, you need a secure shell daemon on your server.

It is beyond the scope of this documentation to detail the installation and setup of a secure shell, however you can find further information in the Subversion / TortoiseSVN SSH HowTo [http://tortoisesvn.berlios.de/faq/index.php?action=artikel&cat=8&id=30&artlang=en]

Further information about svnserve can be found in the SVN book [http://svnbook.red-bean.com].
Chapter 4. The Repository

No matter which protocol you use to access your repositories, you always need to create at least one repository. This can either be done with the Subversion command line client or with TortoiseSVN.

If you haven’t created a Subversion repository yet, it’s time to do that now.

4.1. Repository Creation

You can create a repository with the FSFS backend or with the old but stable Berkeley Database (BDB) format. The FSFS format is faster and it now works on network shares and Windows 98 without problems. The BDB format is more stable because it has been tested longer. Read Chapter 5, Repository Data-Stores [http://svnbook.red-bean.com/en/1.1/ch05.html#svn-ch-5-sect-1.2.A] in the SVN book for more information.

4.1.1. Creating a Repository with the Command Line Client

1. Create an empty folder with the name SVN (e.g. D:\SVN\), which is used as root for all your repositories.
2. Create another folder MyNewRepository inside D:\SVN\.
3. Open the command prompt (or DOS-Box), change into D:\SVN\ and type

   `svnadmin create --fs-type bdb MyNewRepository`

   or

   `svnadmin create --fs-type fsfs MyNewRepository`

   Now you’ve got a new repository located at D:\SVN\MyNewRepository.

4.1.2. Creating The Repository With TortoiseSVN

![TortoiseSVN Menu]

Select "Create Repository here..." to create a new repository.
Figure 4.1. The TortoiseSVN menu for unversioned folders

1. Open the windows explorer
2. Create a new folder and name it e.g. SVNRepository
3. Right-click on the newly created folder and select TortoiseSVN->Create Repository here....

A repository is then created inside the new folder. Don't edit those files yourself!!!. If you get any errors make sure that the folder is empty and not write protected.

4.1.3. Local Access to the Repository

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

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

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

**Warning**

Do not create or access a Berkeley DB repository on a network share. It cannot exist on a remote filesystem. Not even if you have the network drive mapped to a drive letter. If you attempt to use Berkeley DB on a network share, the results are unpredictable - you may see mysterious errors right away, or it may be months before you discover that your repository database is subtly corrupted.

**Tip**

If you really need to access a repository through a network share, create the repository with fsfs format. If you need to provide server access as well, you will need Subversion Server 1.1 or higher.

4.2. Repository Backup

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

The simplest (but not recommended) way is just to copy the repository folder onto the backup medium. However, you have to be absolutely sure that no process is accessing the data. In this context, access means any access at all. A BDB repository is written to even when the operation only appears to require reading, such as getting status. If your repository is accessed at all during the copy, (web browser left open, WebSVN, etc.) the backup will be worthless.

The recommended method is to run

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

The `svnadmin` tool is installed automatically when you install the Subversion command line client. If you are installing the command line tools on a Windows PC, the best way is to download the Windows installer version. It is compressed more efficiently than the `.zip` version, so the download is smaller, and it takes care of setting the paths for you. You can download the latest version of the Subversion command line client from [http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91](http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91).

### 4.3. Hook Scripts

A hook script is a program triggered by some repository event, such as the creation of a new revision or the modification of an unversioned property. Each hook is handed enough information to tell what that event is, what target(s) it's operating on, and the username of the person who triggered the event. Depending on the hook's output or return status, the hook program may continue the action, stop it, or suspend it in some way. Please refer to the chapter on Hook Scripts [http://svnbook.red-bean.com/en/1.1/ch05s02.html#svn-ch-5-sect-2.1](http://svnbook.red-bean.com/en/1.1/ch05s02.html#svn-ch-5-sect-2.1) in the Subversion Book for full details about the hooks which are implemented.

Sample hook scripts can be found in the `hooks` directory of the repository. These sample scripts are suitable for Unix/Linux servers but need to be modified if your server is Windows based. The hook can be a batch file or an executable. The sample below shows a batch file which might be used to implement a pre-revprop-change hook.

```bash
rem Only allow log messages to be changed.
if "%4" == "svn:log" exit 0
echo Property '%4' cannot be changed >&2
exit 1
```

Note that anything sent to stdout is discarded. if you want a message to appear in the Commit Reject dialog you must send it to stderr. In a batch file this is achieved using `>&2`

### 4.4. Checkout Links

If you want to make your Subversion repository available to others you may want to include a link to it from your website. One way to make this more accessible is to include a checkout link for other TSVN users.

When you install TortoiseSVN, it registers a new `tsvn:` protocol. When a TSVN user clicks on such a link, the checkout dialog will open automatically with the repository URL already filled in.

To include such a link in your own html page, you need to add code which looks something like this:

```html
<a href="tsvn:https://svn.collab.net/repos/tortoisesvn/trunk"></a>
```

Of course it would look even better if you included a suitable picture. You can use the [TortoiseSVN logo](https://svn.collab.net/repos/tortoisesvn/trunk/doc/logo/TortoiseCheckout.png) or you can provide your own image.

```html
<a href="tsvn:https://svn.collab.net/repos/tortoisesvn/trunk">
<img src="TortoiseCheckout.png"></a>
```
Chapter 5. Daily Use Guide

This document describes day to day usage of the TortoiseSVN client. It is not an introduction to version control systems, and not an introduction to Subversion (SVN). It is more like a place you may turn to when you know approximately what you want to do, but don't quite remember how to do it.

If you need an introduction to version control with Subversion, then we recommend you read the fantastic book: Subversion: The Definitive Guide [http://svnbook.red-bean.com/].

This document is also a work in progress, just as TortoiseSVN and Subversion are. If you find any mistakes, please report them to the mailing list so we can update the documentation. Some of the screenshots in the Daily Use Guide (DUG) might not reflect the current state of the software. Please forgive us. We're working on TortoiseSVN in our free time.

• You should have installed TortoiseSVN already.
• You should be familiar with version control systems.
• You should know the basics of Subversion.
• You should have set up a server and/or have access to a Subversion repository.

5.1. Getting Started

5.1.1. Context Menus
Figure 5.1. Context menu for a directory under version control

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.

5.1.2. Drag and Drop
Other commands are available as drag handlers, when you right drag files or folders to a new location inside working copies or when you right drag a non-versioned file or folder into a directory which is under version control.

### 5.1.3. Authentication

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

![Authentication Dialog](image)

**Figure 5.3. Authentication Dialog**

Enter your username and password. The checkbox will make TortoiseSVN store the credentials in Subversion's default directory: $APPDATA\Subversion\auth. One file for each server that you access. If you want to make Subversion and TortoiseSVN forget your credentials, you have to delete the corresponding file(s).

For more information on how to set up your server for authentication and access control, refer to Chapter 3, *Setting Up A Server*

---

**Tip**

If you have to authenticate against a Windows NT domain, enter your username including the domain name, like: `MYDOMAIN/johnd`. 
5.2. Importing Data Into A Repository

5.2.1. Repository Layout

Before you import your data into the repository you should first think about how you want to organize your data. If you use one of the recommended layouts you will later have it much easier.

There are some standard, recommended ways to organize a repository. Most people create a trunk directory to hold the "main line" of development, a branches directory to contain branch copies, and a tags directory to contain tag copies. If a repository holds only one project, then often people create these top-level directories:

/trunk
/branches
/tags

If a repository contains multiple projects, people often index their layout by branch:

/trunk/paint
/trunk/calc
/branches/paint
/branches/calc
/tags/paint
/tags/calc

...or by project:

/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags

Indexing by project makes sense if the projects are not closely related and each one is checked out individually. For related projects where you may want to check out all projects in one go, or where the projects are all tied together in a single distribution package, it is often better to index by branch. This way you have only one trunk to checkout, and the relationships between the sub-projects is more easily visible.

If you adopt a top level /trunk /tags /branches approach, there is nothing to say that you have to copy the entire trunk for every branch and tag, and in some ways this structure offers the most flexibility.

For unrelated projects you may prefer to use separate repositories. When you commit changes, it is the revision number of the whole repository which changes, not the revision number of the project. Having 2 unrelated projects share a repository can mean large gaps in the revision numbers. The Subversion and TortoiseSVN projects appear at the same host address, but are completely separate repositories allowing independent development, and no confusion over build numbers.

Of course, you're free to ignore these common layouts. You can create any sort of variation, whatever works best for you or your team. Remember that whatever you choose, it's not a permanent commitment. You can reorganize your repository at any time. Because branches and tags are ordinary directories, TortoiseSVN can move or rename them however you wish.

Switching from one layout to another is just a matter of issuing a series of server-side moves; If you don't like the way things are organized in the repository, just juggle the directories around.

So if you haven't already created a basic folder structure inside your repository you should do that now:
1. create a new empty folder on your hard drive

2. create your desired top-level folder structure inside that folder - don't put any files in it yet!

3. import this structure into the repository via a right click on the folder and selecting TortoiseSVN->Import... This will import your temp folder into the repository root to create the basic repository layout.

Note that the name of the folder you are importing does not appear in the repository, only its contents. For example, create the following folder structure:

```
C:\Temp\New\trunk
C:\Temp\New\branches
C:\Temp\New\tags
```

Import C:\Temp\New into the repository root, which will then look like this:

```
/trunk
/branches
/tags
```

You can also use the repository browser to create new folders directly in the repository.

### 5.2.2. Import

Before you import your project into a repository you should:

1. Remove all files which are not needed to build the project (temporary files, files which are generated by a compiler e.g. *.obj, compiled binaries, ...)

2. Organize the files in folders and subfolders. Although it is possible to rename/move files later it is highly recommended to get your project's structure straight before importing!

Now select the top-level folder of your project directory structure in the windows explorer and right click to open the context menu. Select the command TortoiseSVN->Import... which brings up a dialog box:
Figure 5.4. The Import dialog

In this dialog you have to enter the URL of the repository into which you want to import your project.

The import message is used as a log message. Since you usually just started your project a default initial import is filled in by TortoiseSVN.

**Important**

Files or directories which match the `exclude pattern` are *not* imported.

As soon as you press **OK** TortoiseSVN imports the complete directory tree including all files into the repository. As before, the name of the folder you import does not appear in the repository, only the folder contents. The project is now stored in the repository under version control. Please note that the folder you imported is NOT under version control! To get a version-controlled *working copy* you need to do a Checkout of the version you just imported.

### 5.2.3. Special Files

Sometimes you need to have a file under version control which contains user specific data. That means you have a file which every developer/user needs to modify to suit his/her local setup. But versioning such a file is difficult because every user would commit his/her changes every time to the repository.

In such cases we suggest to use *template* files. You create a file which contains all the data your developers will need, add that file to version control and let the developers check this file out. Then, each developer has to *make a copy* of that file and rename that copy. After that, modifying the copy is not a problem anymore.

As an example, you can have a look at TortoiseSVN's build script. It calls a file named `TortoiseVars.bat` which doesn't exist in the repository. Only the file `TortoiseVars.tmpl`. `TortoiseVars.tmpl` is the template file which every developer has to create a copy from and re-
name that file to `TortoiseVars.bat`. Inside that file, we added comments so that the users will see which lines they have to edit and change according to their local setup to get it working.

So as not to disturb the users, we also added the file `TortoiseVars.bat` to the ignore list of its parent folder, i.e. we've set the Subversion property `svn:ignored` to include that filename. That way it won't show up as unversioned on every commit.

### 5.2.4. Referenced Projects

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 subdirectories to come from different locations in a repository, or perhaps from different repositories altogether. If you want every user to have the same layout, you can define the `svn:externals` properties.

Let's say you check out a working copy of `/project1` to `D:\dev\project1`. Select the folder `D:\dev\project1`, right click and choose `Windows Menu->Properties` from the context menu. The Properties Dialog comes up. Then go to the Subversion tab. There, you can set properties. Select the `svn:externals` property from the combobox and write in the edit box the repository url in the format `name url`. For example:

```
subversion http://svn.collab.net/repos/svn/trunk
```

Now click `Set` and commit your changes. If a user updates their working copy the external project is checked out, too.

If you need more information how TortoiseSVN handles Properties read Section 5.15, “Project Settings”.

### 5.3. Checking Out A Working Copy

To obtain a working copy you need to do a `checkout` from a repository.

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:

![Checkout dialog](image)

**Figure 5.5. The Checkout dialog**
If you enter a folder name that does not yet exist, then a directory with that name is created.

**Important**

You can only check out into an empty directory. If you want to check out a previously imported sourcetree, Subversion will throw an error message. You will have to check out into a different directory or delete the existing sourcetree first.

It is recommended that you check out only the trunk part of the directory tree. If you specify the parent path of the directory tree in the URL then you might end up with a full hard disk since you will get a copy of the entire repository tree including every branch and tag of your project!

**Exporting**

Sometimes you may want a working copy without any of those .svn directories, e.g. to create a zipped tarball of your source. Instead of making a copy and then delete all those .svn directories manually, TortoiseSVN offers the command TortoiseSVN- >Export... If you execute this command on your working copy you'll be asked for a place to save the clean working copy without the .svn folders. Also, you can specify if you want to export all files or just the versioned files.

Another way to do an export is by right-dragging a working copy folder to another location and choose Context Menu->Subversion export here or Context Menu->Subversion export all here

### 5.4. Sending Your Changes To The Repository

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.

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....
Figure 5.6. The Commit dialog

The commit dialog will show you every changed file, including added, deleted and unversioned files. If you don’t want a changed file to be committed, just uncheck that file. If you want to include an unversioned file, just check that file to add it to the commit.

Commit files or folders?

When you commit files, the commit dialog shows only the files you have selected. When you commit a folder the commit dialog will select the changed files automatically. If you forget about a new file you created, committing the folder will find it anyway. Committing a folder does *not* mean that every file gets marked as changed; It just makes your life easier by doing more work for you.

Many unversioned files in the commit dialog

If you think that the TSVN commit dialog shows you too many unversioned (e.g. com-
piler generated or editor backup) files, there are several ways to handle this. You can:

• add the file (or a wildcard extension) to the list of files to exclude on the settings page. This will affect every working copy you have.

• add the file to the svn:ignore list using TortoiseSVN->Add to ignore list
  This will only affect the directory on which you set the svn:ignore property.
  Using the SVN Property Dialog, you can alter the svn:ignore property for a directory.

Read Section 5.11, “Ignoring Files And Directories” for more information.

Doubleclicking on any modified file in the commit dialog will launch the external diff tool to show your changes.

After pressing OK, a dialog appears displaying the progress of the commit.

![SVN Commit... Finished!](image)

<table>
<thead>
<tr>
<th>Action</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified</td>
<td>TortoiseSVN/doc/images/ContextMenuDirControl.png</td>
</tr>
<tr>
<td>Modified</td>
<td>TortoiseSVN/doc/images/ContextMenuDirNoControl.png</td>
</tr>
<tr>
<td>Modified</td>
<td>TortoiseSVN/doc/images/ContextMenuFileControl.png</td>
</tr>
<tr>
<td>Sending Content</td>
<td>C:\TortoiseSVN/doc/images/ContextMenuDirNoControl.png</td>
</tr>
<tr>
<td>Sending Content</td>
<td>C:\TortoiseSVN/doc/images/ContextMenuFileNoControl.png</td>
</tr>
<tr>
<td>Sending Content</td>
<td>C:\TortoiseSVN/doc/images/ContextMenuDirControl.png</td>
</tr>
<tr>
<td>Sending Content</td>
<td>C:\TortoiseSVN/doc/images/ContextMenuFileControl.png</td>
</tr>
<tr>
<td>Completed</td>
<td>At revision: 1575</td>
</tr>
</tbody>
</table>

**Figure 5.7. The Progress dialog showing a commit in progress**

The progress dialog uses colour coding to highlight different commit actions

- Blue
  - Committing a modification or a new item.

- Dark red
  - Committing a deletion or a replacement.

- Black
  - All other items.

**Special Folder Properties**

There are several special folder properties which can be used to help give more control over the formatting of commit log messages and the language used by the spellchecker module. Read Section 5.15, “Project Settings” for further information.
Integration with Bugtracking Tools

If you have activated the bugtracking system, you could set one or more Issues in the Bug-ID / Issue-Nr: Textfield. Several Issues should be comma separated. Learn more Section 5.24, “Integration with Bugtracking Systems / Issue trackers”.

5.5. Update Your Working Copy With Changes From Others

Figure 5.8. Progress dialog showing finished update

Periodically, you should ensure that changes done by others get incorporated in your local working copy. The process of getting changes from the server to your local copy is known as updating. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. To update, select the files and/or directories you want, right click and select TortoiseSVN > Update in the explorer context menu. A window will pop up displaying the progress of the update as it runs. Changes done by others will be merged into your files, keeping any changes you may have done to the same files. The repository is not affected by an update.

The progress dialog uses colour coding to highlight different update actions

- Blue
  - New item added to your WC.

- Dark red
  - Redundant item deleted from your WC.

- Green
  - Changes from repository successfully merged with your local changes.

- Bright red
  - Changes from repository merged with local changes, resulting in conflicts which you need to resolve.

- Black
  - All other items.

If you get any conflicts during an update (this can happen if others changed the same lines in the same file as you did and those changes don't match) then the dialog shows those conflicts in red.
You can double click on these lines to start the external merge tool to resolve the conflicts.

TortoiseSVN also allows you to update your working copy to a specific revision, not only to the most recent one. This command is called TortoiseSVN->Update to Revision... and first opens a dialog where you can enter the required revision. Say e.g. your working copy is at revision 100, but you want your working copy to reflect the state which it had in revision 50 - then simply update to revision 50. In that dialog, you can also decide if you only want to update the current folder only (without all the subfolders) and/or if you want to ignore any external projects in the update (i.e. projects referenced using svn:externals).

**Caution**

If you update a file or folder to a specific revision, you should not make changes to those files. You will get out of date error messages when you try to commit them! If you want to undo changes to a file and start afresh from an earlier revision, you should use one of the following commands instead. The Context Menu->Revert changes from this revision command from the log dialog will undo only those changes made in the selected revision. Changes made after that revision will be retained. For reverting multiple revisions, you should use the TortoiseSVN->Merge... menu command, where you can specify the range of revisions you want to undo.

**Update to Revision** can occasionally be useful to see what your project looked like at some earlier point in its history. But in general, updating individual files to an earlier revision is not a good idea as it leaves your working copy in an inconsistent state. If the file you are updating has changed name, you may even find that the file just disappears from your working copy because no file of that name existed in the earlier revision. If you simply want a local copy of an old version of a file it is better to use the Context Menu->Save revision to... command from the log dialog for that file.

**Multiple Files/Folders**

If you select multiple files and folders in the explorer and then select Update, all of those files/folders are updated one by one. TortoiseSVN makes sure that all files/folders which are from the same repository are updated to the exact same revision! Even if between those updates another commit occurred.

**Local File Already Exists**

Sometimes when you try to update, the update fails with a message to say that there is already a local file of the same name. This typically happens when Subversion tries to checkout a newly versioned file, and finds that an unversioned file of the same name already exists in your working folder. Subversion will never overwrite an unversioned file - it might contain something you are working on, which coincidentally has the same filename as another developer has used for his newly committed file.

If you get this error message, the solution is simply to rename the local unversioned file. After completing the update, you can check whether the renamed file is still needed.

If you keep getting error messages, use TortoiseSVN->Check for Modifications instead to list all the problem files. That way you can deal with them all at once.

### 5.6. Resolving Conflicts

Once in a while, you will get a conflict when you update your files from the repository. A conflict occurs when two or more developers have changed the same few lines of a file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and search for lines starting with the string
The conflicting area is marked like this:

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

Also, for every conflicted file Subversion places three additional files in your directory:

filename.ext.mine
This 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.

filename.ext.rOLDREV
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.

filename.ext.rNEWREV
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.

You can either launch an external merge tool / conflict editor with TortoiseSVN->Edit Conflicts or you can use any other editor to manually resolve the conflict. You should decide what the code should look like, do the necessary changes and save the file.

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.

5.7. Getting Status Information

While you are working on your working copy you often need to know which files you have changed/added/removed or renamed, or even which files got changed and committed by others.

5.7.1. Icon Overlays
Now that you have checked out a working copy from a Subversion repository you can see your files in the windows explorer with changed icons. This is one of the reasons why TortoiseSVN is so popular. TortoiseSVN adds a so called overlay icon to each file icon which overlaps the original file icon. Depending on the Subversion status of the file the overlay icon is different.

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. Read-only 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.

Unlike TortoiseCVS (the CVS shell integration) no overlay icon for unversioned files is shown. We do this because the number of icon overlays are limited system wide and should be used economically.

In fact, you may find that not all of these icons are used on your system. This is because the number of overlays allowed by Windows is limited to 15. Windows uses 4 of those, and the remaining 11 can be used by other applications. If you are also using TortoiseCVS, then there are not enough overlay slots available, so TortoiseSVN tries to be a “Good Citizen (TM)” and limits its use of overlays to give other apps a chance.

- Normal, Modified and Conflicted are always loaded and visible.
- Deleted is loaded if possible, but falls back to Modified if there are not enough slots.
- ReadOnly is loaded if possible, but falls back to Normal if there are not enough slots.
- Locked is only loaded if there are less than 13 overlays already loaded. It falls back to Normal if there are not enough slots.
- Added is only loaded if there are less than 14 overlays already loaded. It falls back to Modified if there are not enough slots.

5.7.2. TortoiseSVN Columns In Explorer

The same information which is available from the icon overlays (and much more) can be displayed as additional columns in Explorer's Detailed View.

Simply right click on one of the headings of a column, choose More... from the context menu displayed. A dialog will appear where you can specify the columns and their order, which is displayed in the “Detailed View”. Scroll down until the entries starting with SVN come into view. Check the ones you would like to have displayed and close the dialog by pressing OK. The columns will be appended to the right of those currently displayed. You can reorder them by drag and drop, or resize them, so that they fit your needs.

Tip

If you want the current layout to be displayed in all your working copies, you may want to make this the default view.

5.7.3. Status Of The Repository
It's often very useful to know which files you have changed and also which files got changed and committed by others. That's where the command TortoiseSVN->Check For Modifications... comes in handy. This dialog will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have.

If you click on the Check Repository then you can also look for changes in the repository. That way you can check before an update if there’s a possible conflict. You can also update selected files from the repository without updating the whole folder.

From the context menu of the dialog you can show a diff of the changes. TortoiseSVN automatically downloads the most recent version from the repository to do the diff.

You can also revert changes in individual files. If you have deleted a file accidentally, it will show up as Missing and you can use Revert to recover it.

**Tip**

If you want a flat view of your working copy, i.e. showing all files and folders at every level of the folder hierarchy, then the Check for Modifications dialog is the easiest way to achieve that. Just check the Show unmodified files checkbox to show all files in your working copy.

### 5.7.4. Viewing Diffs

Often you want to look inside your files, to have a look at what you’ve changed. You can accomplish this by selecting a file which has changed, and selecting Diff from TortoiseSVN's context menu. This starts the external diff-viewer, which will then compare the current file to it’s contents, after the last checkout or update.

**Tip**

Even when not inside a working copy or when you have multiple versions of the file lying around, you can still display diffs:
Select the two files you want to compare in explorer (e.g. using Ctrl and the mouse) and choose Diff from TortoiseSVN's context menu. The file clicked last (the one with the focus, i.e. the dotted rectangle) will be regarded as the later one.

5.8. Revision Log Dialog

For every change you make and commit, you should provide a log message for that change. That way you can later find out what changes you made and why, and you have a detailed log for your development process.

The Revision Log Dialog retrieves all those log messages and shows them to you. The display is divided into 3 panes.

- The top pane shows a list of revisions where changes to the file/folder have been committed. This summary includes the date and time, the person who committed the revision and the start of the log message.
  
  Lines shown in blue indicate that something has been copied to this development line (perhaps from a branch).

- The middle pane shows the full log message for the selected revision.

- The bottom pane shows a list of all files and folders that were changed as part of the selected revision.

But it does much more than that - it provides context menu commands which you can use to get even more information about the project history.

5.8.1. Invoking the Revision Log Dialog
Figure 5.11. The Revision Log Dialog

There are several places from where you can show the Log dialog:

- From the TortoiseSVN context submenu
- From the property page
- From the Progress dialog after an update has finished. Then the Log dialog only shows those revisions which were changed since your last update

5.8.2. Getting Additional Information
Figure 5.12. The Revision Log Dialog Top Pane with Context Menu

The top pane of the Log dialog has a context menu that allows you to

- Compare the selected revision with your working copy. The default Diff-Tool is TortoiseMerge which is supplied with TortoiseSVN. If the log dialog is for a folder, this will show you a list of changed files, and allow you to review the changes made to each file individually.

- View the changes made in the selected revision as a Unified-Diff file. The Unified-Diff operation is faster than invoking the “normal” external diff viewer because TortoiseSVN doesn't have to fetch whole files but only the differences.

- Save the selected revision to a file so you have an older version of that file. This option is only available when you access the log for a file, and it saves a version of that one file only.

- Open the repository browser to examine the selected folder. This option is only available when you access the log for a directory.

- Create a branch/tag from a selected revision. This is useful e.g. if you forgot to create a tag and already committed some changes which weren't supposed to get into that release.

- Update your working copy to the selected revision. Useful if you want to have your working copy reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy will be inconsistent and you will be unable to commit any changes.

- Revert changes from which were made in the selected revision. The changes are reverted in your working copy so this operation does not affect the repository at all! Note that this will undo the changes made in that revision only. It does not replace your working copy with the entire file at the earlier revision. This is very useful for undoing an earlier change when other unrelated changes have been made since.

- Edit the log message or author attached to a previous commit. Read Section 5.8.4, “Changing the Log Message and Author” to find out how this works.

- Search log messages for the text you enter. This searches the log messages that you entered and also the action summaries created by Subversion (shown in the bottom pane). The search is not case sensitive.
If you select two revisions at once (using the usual Ctrl-modifier), the context menu changes and gives you fewer options:

- Compare the two selected revisions using a visual difference tool. The default Diff-Tool is TortoiseMerge which is supplied with TortoiseSVN. If you select this option for a folder, a further dialog pops up listing the changed files and you can diff them one at a time.
- View the differences between the two selected revisions as a Unified-Diff file. This works for files and folders.
- Edit the log message or author as described above.
- Search log messages as described above.

The bottom pane of the Log dialog also has a context menu that allows you to:

- Show differences made in the selected revision for the selected file. This context menu is only available for files shown as Modified.
- Open the selected file, either with the default viewer for that file type, or with a program you choose.
- Revert the changes made to the selected file in that revision.
- View the Subversion properties for the selected item.
• Show the revision log for the selected single file.
• Save the selected revision to a file so you have an older version of that file.

5.8.3. Getting more log messages

The Log dialog does not always show all changes ever made for a number of reasons:

• For a large repository there may be hundreds or even thousands of changes and fetching them all could take a long time. Normally you are only interested in the more recent changes. By default, the number of log messages fetched is limited to 100, but you can change this value in TortoiseSVN->Settings (Section 5.25, “TortoiseSVN’s Settings”),

• When the Stop on copy/rename box is checked, Show Log will stop at the point that the selected file or folder was copied from somewhere else within the repository. This can be useful when looking at branches (or tags) as it stops at the root of that branch, and gives a quick indication of changes made in that branch only.

This option is normally unchecked by default, except when the Show Log dialog is invoked from within the Merge dialog.

Note that Subversion currently implements renaming as a copy/delete pair, so renaming a file or folder will also cause the log display to stop if this option is checked.

If you want to see more log messages, click the Next 100 to retrieve the next 100 log messages. You can repeat this as many times as needed.

If you want to see all log messages right back to revision 1, press Get All.

5.8.4. Changing the Log Message and Author

Sometimes you might want to change a log message you once entered, maybe because there's a spelling error in it or you want to improve the message or change it for other reasons. Or you want to change the author of the commit because you forgot to set up authentication or...

Subversion lets you change both the log message and the author of revisions any time you want. But since such changes can't be undone (those changes are not versioned) this feature is disabled by default. To make this work, you must set up a pre-revprop-change hook. Please refer to the chapter on Hook Scripts [http://svnbook.red-bean.com/en/1.1/ch05s02.html#svn-ch-5-sect-2.1] in the Subversion Book for details about how to do that. Read Section 4.3, “Hook Scripts” to find some further notes on implementing hooks on a Windows machine.

Once you’ve set up your server with the required hooks, you can change both author and log message of any revision, using the context menu from the top pane of the Log dialog.

Warning

Because Subversion's revision properties are not versioned, making modifications to such a property (for example, the svn:log commit message property) will overwrite the previous value of that property forever.

5.8.5. Filtering Log Messages

If you want to restrict the log messages to show only those you are interested in rather than scrolling through a list of hundreds, you can use the filter controls at the top of the Log Dialog. The start and end date controls allow you to restrict the output to a known date range. The search box allows you to show only messages which contain a particular phrase.
Note that these filters act on the messages already retrieved. They do not control downloading of messages from the repository.

5.8.6. Statistical Information

The Statistics button brings up a box showing some interesting information about the revisions shown in the Log dialog. This shows how many authors have been at work, how many commits they have made, progress by week, and much more. Now you can see at a glance who has been working hardest and who is slacking ;-) 

Note that the statistics cover the same period as the Log dialog. If that is only displaying one revision then the statistics will not tell you very much.

5.9. Viewing Differences

One of the commonest requirements in project development is to see what has changed. You might want to look at the differences between two revisions of the same file, or the differences between two separate files. TortoiseSVN provides some builtin tools for this, and also allows you to use your own favourite diff program.

5.9.1. File Differences

Local changes
If you want to see what changes you have made in your working copy, just use the explorer context menu and select TortoiseSVN->Diff.

Difference from a previous revision
If you want to see the difference between a particular revision and your working copy, use the Revision Log dialog, select the revision of interest, then select Compare with working copy from the context menu.

Difference between two previous revisions
If you want to see the difference between two revisions which are already committed, use the Revision Log dialog and select the two revisions you want to compare (using the usual Ctrl-modifier). Then select Compare revisions from the context menu.

All changes made in a commit
If you want to see the changes made to all files in a particular revision in one view, you can use Unified-Diff output (GNU patch format). This shows only the differences with a few lines of context. It is harder to read than a visual file compare, but will show all the changes together. From the Revision Log dialog select the revision of interest, then select Show Differences as Unified-Diff from the context menu.

Difference between files
If you want to see the differences between two different files, you can do that directly in explorer by selecting both files (using the usual Ctrl-modifier). Then from the explorer context menu select TortoiseSVN->Diff.

5.9.2. Folder Differences

The builtin tools supplied with TortoiseSVN do not support viewing differences between directory hierarchies. But if you have an external tool which does support that feature, you can use that instead. In the next section we tell you about some tools which we have used.

5.9.3. External Diff/Merge Tools

If the tools we provide don’t do what you need, try one of the many open-source or commercial programs available. Everyone has their own favourites, and this list is by no means complete, but here are a few that you might consider:
WinMerge
WinMerge [http://winmerge.sourceforge.net/] is a great open-source diff viewer and merge tool.

Perforce WinDiff
Perforce is a commercial RCS, but you can download the diff/merge tool for free. Download the full installer from Perforce [http://www.perforce.com/perforce/loadprog.html].

KDiff3
KDiff3 is a free diff tool which can also handle directories. You can download it from here [http://kdiff3.sf.net/].

ExamDiff
ExamDiff Standard is freeware. It can handle files but not directories. ExamDiff Pro is shareware and adds a number of goodies including directory diff and editing capability. In both flavours, version 3.2 and above can handle unicode. You can download them from PrestoSoft [http://www.prestosoft.com/].

Beyond Compare
Similar to ExamDiff Pro, this is an excellent shareware diff tool which can handle directory diffs and unicode. Download it from Scooter Software [http://www.scootersoftware.com/].

Araxis Merge
Araxis Merge is a useful commercial tool for diff and merging both files and folders. It does three-way comparison in merges and has synchronization links to use if you've changed the order of functions. Download it from Araxis [http://www.araxis.com/merge/index.html].

SciTE
This text editor includes syntax colouring for unified diffs, making them much easier to read. Download it from Scintilla [http://www.scintilla.org/SciTEDownload.html].

Notepad2
Notepad2 is designed as a replacement for the standard Windows Notepad program, and is based on the Scintilla open-source edit control. As well as being good for viewing unified diffs, it is much better than the Windows notepad for most jobs. Download it for free here [http://www.flos-freeware.ch/notepad2.html].

TkDiff
A free diff/merge tool which uses TCL/TK. Download it here [http://www.accurev.com/free/tkdiff/].

CVSConflictEditor
This open source merge tool is buggy, but has potential. Download it here [http://sourceforge.net/projects/conflicteditor/].

Read Section 5.25.4, “External Program Settings” for information on how to set up TortoiseSVN to use these tools.

5.10. Adding New Files And Directories
If you created new files and/or directories during your development process 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 an 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!

Many Adds

You can also use the Add command on already versioned folders. In that case, the add dialog will show you all unversioned files inside that versioned folder. This helps if you have many new files and need them to add at once.

To add files from outside your working copy you can use the drag-and-drop handler:

1. select the files you want to add
2. right-drag them to the new location inside the working copy
3. release the right mouse button
4. select Context Menu->Add files to Subversion here. The files will then be copied to the working copy and added to version control.

5.11. Ignoring Files And Directories
Figure 5.16. Explorer context menu for unversioned files

In most projects you will have files and folders that should not be subject to version control. These might include files created by the compiler, *.obj, *.lst, maybe an output folder used to store the executable. Whenever you commit changes, TSVN shows your unversioned files, which fills up the file list in the commit dialog. Of course you can turn off this display, but then you might forget to add a new source file.

The best way to avoid these problems is to add the derived files to the project's ignore list. That way they will never show up in the commit dialog, but genuine unversioned source files will still be flagged up.

If you right click on a single unversioned file, and select the command TortoiseSVN->Add to Ignore List from the context menu, a submenu appears allowing you to select just that file, or all files with the same extension. If you select multiple files, there is no submenu and you can only add those specific files/folders.

If you want to remove one or more items from the ignore list, right click on those items and select TortoiseSVN->Remove from Ignore List You can also access a folder's svn:ignore property directly. Read Section 5.15, “Project Settings” for more information.

The Global Ignore List

Another way to ignore files is to add them to the global ignore list. The big difference here is that the global ignore list is a client property. It applies to all Subversion projects, but on the client PC only. In general it is better to use the svn:ignore property where possible, because it can be applied to specific project areas, and it works for everyone who checks out the project. Read Section 5.25.1, “General Settings” for more information.

5.12. Deleting, Renaming And Moving

Unlike CVS, Subversion allows renaming and moving of files and folders. So there are menu entries for delete and rename in the TortoiseSVN submenu.
If you delete a file/directory using TSVN, the file is removed from your working copy and marked for deletion. The file's parent folder shows a "deleted" icon overlay. You can always get the file back, if you call TortoiseSVN->Revert on the parent folder.

If you want to move files inside a working copy, use the drag-and-drop handler again:

1. select the files or directories you want to move
2. right-drag them to the new location inside the working copy
3. release the right mouse button
4. in the popup menu select Context Menu->Move files in Subversion to here

If a file is deleted via the explorer but not removed from version control (i.e. you delete a file as usual), the commit dialog shows those files and lets you remove them from version control too before the commit. So it doesn't really matter, if you forget to delete the file with TortoiseSVN as you are supposed to do.

**Commit the parent folder**

Since renames and moves are done as a delete followed by an add you must commit the parent folder of the renamed/moved file. Only that way the deleted part of the re-
name/move will show up in the commit dialog. If you don’t commit the removed part of the rename/move, it will stay behind in the repository and an update of your coworkers won’t remove the old file.

### Getting a deleted file or folder back

If you have deleted a file or a folder and already committed that delete operation to the repository, then a normal TortoiseSVN->Revert can’t bring it back anymore. But the file or folder is not lost at all. If you know the revision the file or folder got deleted (if you don’t, use the log dialog to find out) open the repository browser and switch to that revision. Then select the file or folder you deleted, right-click and select Context Menu->Copy to... as the target for that copy operation select the path to your working copy.

### 5.13. Undo Changes

If you want to undo all changes you made in a file since the last update you need to select the file, right click to pop up the context menu and then select the command TortoiseSVN->Revert A dialog will pop up showing you the files that you’ve changed and can revert. Select those you want to revert and click on OK.

![Figure 5.18. Revert dialog](image)

55
Undoing Changes which have been Committed

Revert will only undo your local changes. It does not undo any changes which have already been committed. If you want to undo all the changes which were committed in a particular revision, read Section 5.8, “Revision Log Dialog” for further information.

5.14. Cleanup

If a subversion command cannot complete successfully, perhaps due to server problems, your working copy can be left in an inconsistent state. In that case you need to use TortoiseSVN->Cleanup on the folder. It is a good idea to do this at the top level of the working copy.

Cleanup has another useful side effect. If a file date changes but its content doesn't, subversion cannot tell whether it has really changed except by doing a byte-by-byte comparison with the pristine copy. If you have a lot of files in this state it makes acquiring status very slow, which will make many dialogs slow to respond. Executing a Cleanup on your working copy will repair these “broken” timestamps and restore status checks to full speed.

Use Commit Timestamps

Some earlier releases of Subversion were affected by a bug which caused timestamp mismatch when you check out with the Use commit timestamps option checked. Use the Cleanup command to speed up these working copies.

5.15. Project Settings
Figure 5.19. Subversion property page

Sometimes you want to have a more detailed information about a file/directory than just the icon overlay. You can get all the information Subversion provides in the explorer properties dialog. Just select the file or directory and select Windows Menu->properties in the context menu (note: this is the normal properties menu entry the explorer provides, not in the TortoiseSVN submenu!). In the properties dialog box TortoiseSVN has added a new property page for files/folders under Subversion control.

On the Subversion page you can see all relevant information about the selected file/directory. In ad-
dition you can read and set the Subversion properties. You can add your own properties, or some properties with a special meaning in Subversion. These begin with svn:svn:externals is such a property; see how to handle externals in Section 5.2.4, “Referenced Projects”. For more information about properties in Subversion see the Subversion Manual [http://svnbook.red-bean.com/en/1.1/ch07s02.html#svn-ch-7-sect-2.3].

To set a property, select the required property name from the combo box, then type in a value in the field below. Properties which take multiple values, such as an ignore list, can be entered on multiple lines. Click on Set to add that property to the list.

If you want to apply a property to many items at once, select the files/folders in explorer, then select Windows Menu->properties.

If you want to apply the property to every file and folder in the hierarchy below the current folder, check the Recursive checkbox.

Some properties, for example svn:needs-lock, can only be applied to files, so the property name doesn't appear in the drop down list for folders. You can still apply such a property recursively to all files in a hierarchy, but you have to type in the property name yourself.

If you wish to edit an existing property, first select that property from the list of existing properties, so that it appears in the edit area. Then proceed as for a new property.

If you wish to remove an existing property, select that property from the list of existing properties, so that it appears in the edit area. Then click on Remove.

TortoiseSVN has a few special properties of its own, and these begin with tsvn:

• tsvn:logminsize sets the minimum length of a log message for a commit. If you enter a shorter message than specified here, the commit is disabled. This feature is very useful for reminding you to supply a proper descriptive message for every commit. If this property is not set, or the value is zero, empty log messages are allowed.

• tsvn:lockmsgminsize sets the minimum length of a lock message. If you enter a shorter message than specified here, the lock is disabled. This feature is very useful for reminding you to supply a proper descriptive message for every lock you get. If this property is not set, or the value is zero, empty lock messages are allowed.

• tsvn:logwidthmarker is used with projects which require log messages to be formatted with some maximum width (typically 80 characters) before a line break. Setting this property to a non-zero will do 2 things in the log message entry dialog: it places a marker to indicate the maximum width, and it disables word wrap in the display, so that you can see whether the text you entered is too long. Note: this feature will only work correctly if you have a fixed-width font selected for log messages.

• tsvn:logtemplate is used with projects which have rules about log message formatting. The property holds a multi-line text string which will be inserted in the commit message box when you start a commit. You can then edit it to include the required information. Note: if you are also using tsvn:logminsize, be sure to set the length longer than the template or you will lose the protection mechanism.

• In the Commit dialog you have the option to paste in the list of changed files, including the status of each file (added, modified, etc). tsvn:logfilelistenglish defines whether the file status is inserted in english or in the localized language. If the property is not set, the default is true.

• TortoiseSVN can use spell checker modules which are also used by OpenOffice and Mozilla. If you have those installed this property will determine which spell checker to use, i.e. in which language the log messages for your project should be written. tsvn:projectlanguage sets the language module the spell checking engine should use when you enter a log message. You can find the values for your language on this page: MSDN: Language Identifiers [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/intl/nls_238z.asp] true.
Some tsvn: properties require a true/false value. TSVN also understands yes as a synonym for true and no as a synonym for false.

Set the tsvn: properties on folders

These tsvn: properties must be set on folders for the system to work. When you commit a file or folder the properties are read from that folder. If the properties are not found there, TortoiseSVN will search upwards through the folder tree to find them until it comes to an unversioned folder, or the tree root (e.g. C:\) is found. If you can be sure that each user checks out only from e.g trunk/ and not some subfolder, then it is sufficient to set the properties on trunk/. If you can’t be sure, you should set the properties recursively on each subfolder. A property setting deeper in the project hierarchy overrides settings on higher levels (closer to trunk/).

For tsvn: properties only you can use the Recursive checkbox to set the property to all subfolders in the hierarchy, without also setting it on all files.

Commit properties

Subversion properties are versioned. After you change or add a property you have to commit your changes.

Conflicts on properties

If there’s a conflict on committing the changes, because another user has changed the same property, Subversion generates a .prej file. Delete this file after you have resolved the conflict.

Automatic property setting

You can configure Subversion to set properties automatically on files and folders when they are added to the repository. Read Section 5.25, “TortoiseSVN's Settings” for further information.

TortoiseSVN can integrate with some bugtracking tools. This uses properties, which start with bugtraq:. Read Section 5.24, “Integration with Bugtracking Systems / Issue trackers” for further information.

5.16. Branching / Tagging

One of the features of version control systems is the ability to isolate changes onto a separate line of development. 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).

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.
5.16.1. Creating a Branch or Tag

If you have imported your project with the recommended directory structure, creating a branch or tag version is very simple:

![Figure 5.20. The Branch/Tag Dialog](image)

**Figure 5.20. The Branch/Tag Dialog**

Select the folder in your working copy which you want to copy to a branch or tag, then select the command TortoiseSVN->Branch/Tag....

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
```
If you can’t remember the naming convention you used last time, click the button on the right to open the repository browser so you can view the existing repository structure.

Now you have to select the source of the copy. Here you have three options:

**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 WC, 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 WC back to the repository if it does not exist there already.

Press **OK** to commit the new copy to the repository. Don’t forget to supply a log message. Note that the copy is created **inside the repository**.

Note that creating a Branch or Tag does **not** affect your working copy. Even if you copy your WC, those changes are committed to the new branch, not to the trunk, so your WC may still be marked as modified.

### 5.16.2. To Checkout or to Switch...

...that is (not really) the question. While a checkout checks out everything from the desired branch into your working directory, *TortoiseSVN->Switch...* only transfers the changed data to your working copy. Good for the network load, good for your patience. :-)

To be able to work with your freshly generated copy you have several ways to handle it. You can:

- *TortoiseSVN->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** 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.

  In the next dialog enter the URL of the branch you just created. Select the **Head Revision** radio button and click on **OK**. Your working copy is switched to the new branch/tag.

  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).
Although Subversion itself makes no distinction between tags and branches, the way they are typically used differs a bit.

- Tags are typically used to create a static snapshot of the project at a particular stage. As such they not normally used for development - that's what branches are for, which is the reason we recommended the /trunk /branches /tags repository structure in the first place. Working on a 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 to commit to a path in the repository which contains /tags/, TortoiseSVN will warn you.

- 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.

- If you modify a working copy created from a branch and commit, then all changes go to the new branch and not the trunk. Only the modifications are stored. The rest remains a cheap copy.

### 5.17. Merging

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.

It is important to understand how branching and merging works in Subversion before you start using it, as it can become quite complex. It is highly recommended that you read the chapter Branching and Merging [http://svnbook.red-bean.com/en/1.1/ch04.html] in the Subversion Book [http://svnbook.red-bean.com/] which gives a full description, and many examples of how it is used.

An important point to remember is that Merge is closely related to Diff. The merge process works by generating a list of differences between two points in the repository, and applying those differences to your working copy. For example if you want to merge the changes made in revision N then you have to compare revision N with revision (N-1). Novices often ask “Why do I have to subtract 1 from the start revision.” Think of the underlying Diff process and it will become clearer. TO make this easier, when you use Show Log to select a range of revisions to merge, TortoiseSVN makes this adjustment for you automatically.
In general it is a good idea to perform a merge into an unmodified working copy. If you have made other changes in your WC, 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.

There are two common use cases for merging which are handled in slightly different ways, as described below.

5.17.1. Merging 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.

![Figure 5.22. The Merge Dialog](image)

To merge revisions you need to go to a working copy of the branch in which you want to receive the changes, often the trunk. Select TortoiseSVN->Merge... from the context menu.

1. In the From: field enter 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. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

2. Because you are porting a range of revisions from the same branch into your working copy, make sure the Use "From:" URL checkbox is checked.
3. In the From Revision field enter the start revision number. This is the revision before the changes you want to merge. Remember that Subversion will create a diff file in order to perform the merge, so the start point has to be just before the first change you are interested in. For example, your log messages may look something like this:

<table>
<thead>
<tr>
<th>Rev</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Working on MyBranch</td>
</tr>
<tr>
<td>38</td>
<td>Working on trunk</td>
</tr>
<tr>
<td>37</td>
<td>Working on MyBranch</td>
</tr>
<tr>
<td>36</td>
<td>Create branch MyBranch</td>
</tr>
<tr>
<td>35</td>
<td>Working on trunk</td>
</tr>
<tr>
<td>34</td>
<td>Working on trunk</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you now want to merge all the changes from MyBranch into the trunk you have to choose 36 as the From Revision, not 37 as you might think. If you select revision 37 as the start point, then the difference engine compares the end point with revision 37, and will miss the changes made in revision 37 itself. If that sounds complicated, don't worry, there is an easier way in TortoiseSVN ...

The easiest way to select the range of revisions you need is to click on Show Log, as this will list recent changes with their log comments. If you want to merge the changes from a single revision, just select that revision. If you want to merge changes from several revisions, then select that range (using the usual Shift-modifier). Click on OK and the revision numbers of the From revision and To revision in the Merge dialog will both be filled in for you.

If you have already merged some changes from this branch, hopefully you will have made a note of the last revision merged in the log message when you committed the change. In that case, you can use Show Log for the Working Copy to trace that log message. Use the end point of the last merge as the start point for this merge. For example, if you have merged revisions 37 to 39 last time, then the start point for this merge should be revision 39.

4. If you have not used Show Log to select the revision range, then you will need to set the To Revision manually. Enter the last revision number in the range you want to merge. Often this will be the HEAD revision, although it doesn't need to be - you may just want to merge a single revision.

If other people may be committing changes then be careful about using the HEAD revision. It may not refer to the revision you think it does if someone else made a commit after your last update.

5. Click OK to complete the merge.

The merge is now complete. It's a good idea to have a look at the merge and see if it's as expected. Merging is usually quite complicated. Conflicts often arise if the branch has drifted far from the trunk.

When you have tested the changes and come to commit this revision, your commit log message should always include the revision numbers which have been ported in the merge. If you want to apply another merge at a later time you will need to know what you have already merged, as you do not want to port a change more than once. Unfortunately merge information is not stored by Subversion. For more information about this, refer to Tracking Merges Manually [http://svnbook.red-bean.com/en/1.1/ch04s03.html#svn-ch-4-sect-3.2.1] in the Subversion Book [http://svnbook.red-bean.com/]

Branch management is important. If you want to keep this branch up to date with the trunk, you should be sure to merge often so that the branch and trunk do not drift too far apart. Of course, you should still avoid repeated merging of changes, as explained above.

**Important**

Subversion can't merge a file with a folder and vice versa - only folders to folders and
files to files. If you click on a file and open up the merge dialog, then you have to give a path to a file in that dialog. If you select a folder and bring up the dialog, then you must specify a folder url for the merge.

5.17.2. Merging Two Different Trees

This method covers the case when you have made a feature branch as discussed in the Subversion book. All trunk changes have been ported to the feature branch, week by week, and now the feature is complete you want to merge it back into the trunk. Because you have kept the feature branch synchronized with the trunk, the latest versions of branch and trunk will be absolutely identical except for your branch changes. So in this special case, you would merge by comparing the branch with the trunk.

To merge the feature branch back into the trunk you need to go to a working copy of the trunk. Select TortoiseSVN->Merge... from the context menu.

1. In the From: field enter 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.

2. Because you are comparing two different trees, make sure the Use "From:" URL checkbox is not checked.

3. In the To: field enter the full folder url of the feature branch.

4. In both the From Revision field and the To Revision field, enter 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. Note that in this case you are not selecting a range of revisions, so the revision you select there is what will actually appear in the Revision field.

5. Click OK to complete the merge.

In this case you will not need the feature branch again because the new feature is now integrated into the trunk. The feature branch is redundant and can be deleted from the repository if required.

5.17.3. Previewing Merge Results

If you are uncertain about the merge operation, you may want to preview what will happen before you allow it to change your working copy. There are two additional buttons to help you.

Unified Diff creates the diff file (remember that merge is based on diff) and shows you which lines will be changed in your working copy files. As this is a unified diff (patch) file it is not always easy to read out of context, but for small scale changes it is often helpful.

Dry Run performs 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 those areas where conflicts will occur.

5.18. Locking

Subversion generally works best without locking, using the “Copy-Modify-Merge” methods described earlier in Section 2.2.3, “The Copy-Modify-Merge Solution”. However there are a few instances when you may need to implement some form of locking policy.
• You are using “unmergeable” files, for example, graphics files. If two people change the same file, merging is not possible, so one of you will lose their changes.

• Your company has always used a locking VCS in the past and there has been a management decision that “locking is best”.

Firstly you need to ensure that your Subversion server is upgraded to at least version 1.2. Earlier versions do not support locking at all. If you are using file:// access, then of course only your client needs to be updated.

5.18.1. How Locking Works in Subversion

By default, nothing is locked and anyone who has commit access can commit changes to any file at any time. Others will update their working copies periodically and changes in the repository will be merged with local changes.

If you Get a Lock on a file, then only you can commit that file. Commits by all other users will be blocked until you release the lock. A locked file cannot be modified in any way in the repository, so it cannot be deleted or renamed either, except by the lock owner.

However, other users will not necessarily know that you have taken out a lock. Unless they check the lock status regularly, the first they will know about it is when their commit fails, which in most cases is not very useful. To make it easier to manage locks, there is a new Subversion property `svn:needs-lock`. When this property is set (to any value) on a file, whenever the file is checked out or updated, the local copy is made read-only unless that working copy holds a lock for the file. This acts as a warning that you should not edit that file unless you have first acquired a lock. Files which are versioned and read-only are marked with a special overlay in TortoiseSVN to indicate that you need to acquire a lock before editing.

Locks are recorded by working copy location as well as by owner. If you have several working copies (at home, at work) then you can only hold a lock in one of those working copies.

If one of your co-workers acquires a lock and then goes on holiday without releasing it, what do you do? Subversion provides a means to force locks. Releasing a lock held by someone else is referred to as Breaking the lock, and forcibly acquiring a lock which someone else already holds is referred to as Stealing the lock. Naturally these are not things you should do lightly if you want to remain friends with your co-workers.

Locks are recorded in the repository, and a lock token is created in your local working copy. If there is a discrepancy, for example if someone else has broken the lock, the local lock token becomes invalid. The repository is always the definitive reference.

5.18.2. Getting a Lock

Select the file(s) in your working copy for which you want to acquire a lock, then select the command TortoiseSVN->Get Lock....
5.18.3. Releasing a Lock

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

To release a lock manually, select the file(s) in your working copy for which you want to release the lock, then select the command TortoiseSVN->Release Lock There is nothing further to enter so TortoiseSVN will contact the repository and release the locks.

5.18.4. Checking Lock Status
Locally held lock tokens show up immediately. To check for locks held by others (and to see if any of your locks are broken or stolen) you need to click on Check Repository.

From the context menu here, you can also get and release locks, as well as breaking and stealing locks held by others.

Avoid Breaking and Stealing Locks

If you break or steal someone else's lock without telling them, you could potentially cause loss of work. If you are working with unmergeable file types and you steal someone else's lock, once you release the lock they are free to check in their changes and overwrite yours. Subversion doesn't lose data, but you have lost the team-working protection that locking gave you.

5.18.5. Making Non-locked Files Read-Only

As mentioned above, the most effective way to use locking is to set the svn:needs-lock property on files. Refer to Section 5.15, “Project Settings” for instructions on how to set properties. Files with this property set will always be checked out and updated with the read-only flag set unless your working copy holds a lock.

As a reminder, TortoiseSVN uses a special overlay to indicate this.

If you operate a policy where every file has to be locked then you may find it easier to use Subversion's auto-props feature to set the property automatically every time you add new files. Read Section 5.25, “TortoiseSVN's Settings” for further information.

5.18.6. The Locking Hook Scripts

When you create a new repository with Subversion 1.2 or higher, four hook templates are created in the repository hooks directory. These are called before and after getting a lock, and before and after releasing a lock.
after releasing a lock.

It is a good idea to install a post-lock and post-unlock hook script on the server which sends out an email indicating the file which has been locked. With such a script in place, all your users can be notified if someone locks/unlocks a file. You can find an example hook script hooks/post-lock.tmpl in your repository folder.

You might also use hooks to disallow breaking or stealing of locks, or perhaps limit it to a named administrator. Or maybe you want to email the owner when one of their locks is broken or stolen.

Read Section 4.3, “Hook Scripts” to find out more.

5.19. Creating and Applying Patches

For open source projects (like this one) everyone has read access to the repository, and anyone can make a contribution to the project. 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.

5.19.1. 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... This will produce a single file containing a summary of all the changes you have made since the last update from the repository. If you create a patch file, make some more changes 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 .patch or .diff extension. You are now ready to submit your patch file.

5.19.2. 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 was doc/source/english/chapter1.xml and the first line in the patchfile is Index: english/chapter1.xml then you need to apply the patch to the english folder. However, provided you are in the correct working copy, if you pick the wrong folder level, TSVN 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".

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.

5.20. Who Changed Which Line?

![Blame Dialog](image)

Figure 5.25. The Annotate / Blame Dialog

Sometimes you need to know not only what lines have changed, but also who exactly changed specific lines in a file. That's when the TortoiseSVN->Blame... command, sometimes also referred to as annotate command comes in handy.

This command lists, for every line in a file, the author and the revision the line was changed.

If you're not interested in changes from earlier revisions you can set the revision from which the blame should start. Set this to 1, if you want the blame for every revision.

By default the blame file is viewed using TortoiseBlame, which highlights the different revisions to make it easier to read. If you wish to print or edit the blame file, select Use Text viewer to view blames.

Once you press OK TortoiseSVN starts retrieving the data to create the blame file. Please note: This can take several minutes to finish, depending on how much the file has changed and of course your network connection to the repository. Once the blame process has finished the result is written into a temporary file and you can view the results.
Figure 5.26. TortoiseBlame

TortoiseBlame, which is included with TortoiseSVN, makes the blame file easier to read. When you hover the mouse over a line in the blame info column, all lines with the same revision are shown with a darker background. Lines from other revisions which were changed by the same author are shown with a light background. The colouring may not work as clearly if you have your display set to 256 colour mode.

If you left click on a line, all lines with the same revision are highlighted, and lines from other revisions by the same author are highlighted in a lighter colour. This highlighting is sticky, allowing you to move the mouse without losing the highlights. Click on that revision again to turn off highlighting.

The revision comments are shown in a hint box whenever the mouse hovers over the blame info column.

5.21. The Repository Browser

Sometimes you need to work directly on the repository, without having a working copy. That’s what the Repository Browser is for. What the explorer and the icon overlays are for your working copy is the Repository Browser for the repository.
Figure 5.27. The Repository Browser

With the Repository Browser you can execute commands like copy, move, rename, ... directly on the repository.

On the top of the Repository Browser Window you can enter the URL of the repository and the revision you want to browse. Browsing an older revision is useful if you want to e.g. recover a previously deleted file. Use the Context Menu->Copy To... command to do that and enter the full working copy path to where you want to recover your deleted file.

5.22. Revision Graphs
Sometimes you need to know where branches and tags were taken from the trunk, and the ideal way to view this sort of information is as a graph or tree structure. That's when you need to use Tor- toiseSVN->Revision Graph...

This command analyses the revision history and attempts to create a tree showing the points at which copies were taken, and when branches/tags were deleted. Please note: It can take several minutes to retrieve all the data needed to create the revision graph, depending on how much the

Figure 5.28. A Revision Graph
project has changed and of course your network connection to the repository.

The revision graph shows several types of node:

**Added file/folder**
- Items which have been added, or created by copying another file/folder are shown using a rounded rectangle.

**Deleted file/folder**
- Deleted items eg. a branch which is no longer required, are shown using an octagon (rectangle with corners cut off).

**Normal file/folder**
- All other items are shown using a plain rectangle.

Note that the graph only shows the points at which items were added or deleted. Showing every revision of a project would generate an impossibly large graph for non-trivial cases. For this reason, simple project structures with no adds or deletes will produce an empty revision graph.

The revision date, author and comments are shown in a hint box whenever the mouse hovers over a revision box.

If you select two revisions, you can use the context menu to show the differences between these revisions. You can choose to show differences as at the branch creation points, but usually you will want to show the differences at the branch end points, i.e. at the HEAD revision.

You can view the differences as a Unified-Diff file, which shows all differences in a single file with minimal context. If you opt to Context Menu->Compare Revisions you will be presented with a list of changed files. Double click on a file name to fetch both revisions of the file and compare them using the visual difference tool.

**Caution**
Because Subversion cannot provide all the information required, a certain amount of interpretation is required, which can sometimes give strange results. Nevertheless, the output for the trunk will generally give useful results.

### 5.23. Relocating a working copy

![Relocate Dialog](image)

**Figure 5.29. The Relocate Dialog**

If your repository has for some reason changed it's location (IP/URL). Maybe you're even stuck and can't commit and you don't want to checkout your working copy again from the new location and to
move all your changed data back into the new working copy. TortoiseSVN->Relocate is the command you are looking for. It basically does very little: it scans all "entries" files in the .svn folder and changes the URL of the entries to the new value.

**Warning**

Don't use the relocate command to switch your working copy to a different branch! If you do that, then you will get many unexplainable error messages while updating, committing, ...

The relocate command must *only* be used if the URL of the server changed!

### 5.24. Integration with Bugtracking Systems / Issue trackers

It is very common in Software Development for changes to be related to a specific bug or issue ID. Users of bug tracking systems (issue trackers) would like to associate the changes they make in Subversion with a specific ID in their issue tracker. Most issue trackers therefore provide a pre-commit hook script which parses the log message to find the bug ID with which the commit is associated. This is somewhat error prone since it relies on the user to write the log message properly so that the pre-commit hook script can parse it correctly.

TortoiseSVN can help the user in two ways:

1. When the user enters a log message, a well defined line including the issue number associated with the commit can be added automatically. This reduces the risk that the user enters the issue number in a way the bug tracking tools can't parse correctly.
   Or TortoiseSVN can highlight the part of the entered log message which is recognized by the issue tracker. That way the user knows that the log message can be parsed correctly.

2. When the user browses the log messages, TortoiseSVN creates a link out of each bug ID in the log message which fires up the browser to the issue mentioned.

You can integrate a Bugtracking Tool of your choice in TortoiseSVN. To do this, you have to define some properties, which start with `bugtraq:`. They must be set on Folders: (Section 5.15, “Project Settings”)

There are two ways to integrate TortoiseSVN with issue trackers. One is based on simple strings, the other is based on regular expressions. The properties used by both approaches are:

**bugtraq:url**
Set this property to the url of your bugtracking tool. It must be properly URI encoded and it has to contain `%BUGID%`. `%BUGID%` is replaced with the Issuenumber you entered. This allows TortoiseSVN to display a link in the log dialog, so when you are looking at the revision log you can jump directly to your bugtracking tool. You do not have to provide this property, but then TortoiseSVN shows only the issuenumber and not the link to it. e.g the TortoiseSVN project is using [http://tortoisesvn.tigris.org/issues/show_bug.cgi?id=%BUGID%](http://tortoisesvn.tigris.org/issues/show_bug.cgi?id=%BUGID%)

**bugtraq:warnifnoissue**
Set this to `true`, if you want TortoiseSVN to warn you because of an empty issuenumber text-field. Valid values are `true/false`. *If not defined, `false` is assumed.*

In the simple approach, TortoiseSVN shows the user a separate input field where a bug ID can be entered. Then a separate line is appended/prepended to the log message the user entered.

**bugtraq:message**
*This property activates the Bugtracking System.* If this property is set, then TortoiseSVN will prompt you to enter an issue number when you commit your changes. It's used to add a line at
the end of the logmessage. It must contain %BUGID%, which is replaced with the issuenumber on commit. This ensures that your commit log contains a reference to the issuenumber which is always in a consistent format and can be parsed by your Bugtracking tool to associate the issuenumber with a particular commit. e.g. the TortoiseSVN project is using Issue : %BUGID%, but this depends on your Tool.

**bugtraq:append**
This property defines if the bug-ID is appended (true) to the end of the log message or inserted (false) at the start of the log message. Valid values are true/false. If not defined, true is assumed, so that existing projects don't break.

**bugtraq:label**
This text is shown by TortoiseSVN on the commit dialog to label the edit box where you enter the issuenumber. If it's not set, Bug-ID / Issue-Nr: will be displayed. Keep in mind though that the window will not be resized to fit this label, so keep the size of the label below 20-25 characters.

**bugtraq:number**
If set to true only numbers are allowed in the issuenumber textfield. An exception is the comma, so you can comma separate several numbers. Valid values are true/false. If not defined, true is assumed.

In the approach with regular expressions, TortoiseSVN doesn't show a separate input field but marks the part of the log message the user enters which is recognized by the issue tracker. This is done while the user writes the log message. This also means that the bug ID can be anywhere inside a log message!

---

**Tip**

Even if you don't have an issue tracker with a pre-commit hook parsing your log messages, you still can use this to turn the issues mentioned in your log messages into links!

---

**bugtraq:logregex**
This property contains one or two regular expressions, separated by a newline.

If only one expression is set, then the bare bug ID's must be matched in the groups of the regex string. Example: [Ii]ssue(?:s)? #?\(\d+\)

If two expressions are set, then the first expression is used to find a string which relates to the bug ID but may contain more than just the bug ID (e.g. "Issue #123" or "resolves issue 123"). The second expression is then used to extract the bare bug ID from the string extracted with the first expression. An example:

If you want to catch every pattern "issue #XXX" and "issue #890, #789" inside a log message you could use the following regex strings: [Ii]ssue #\(\d+\)\(, ? #\\'\(\d+\)\)\+ and the second expression as \(\d+\)

If you are unfamiliar with regular expressions, take a look at the online documentation and tutorial at [http://www.regular-expressions.info/](http://www.regular-expressions.info/).

Some tsvn: properties require a true/false value. TSVN also understands yes as a synonym for true and no as a synonym for false.

---

**Set the Properties on Folders**

These properties must be set on folders for the system to work. When you commit a file or folder the properties are read from that folder. If the properties are not found there, TortoiseSVN will search upwards through the folder tree to find them until it
comes to an unversioned folder, or the tree root (e.g. C:) is found. If you can be sure that each user checks out only from e.g trunck/ and not some subfolder, then it's enough if you set the properties on trunck/. If you can't be sure, you should set the properties recursively on each subfolder. A property setting deeper in the project hierarchy overrides settings on higher levels (closer to trunck/).

For tsvn: properties only you can use the Recursive checkbox to set the property to all subfolders in the hierarchy, without also setting it on all files.

This issue tracker integration is not restricted to TortoiseSVN; it can be used with any Subversion client. For more information, read the full Issuetracker Integration Specification [http://svn.collab.net/repos/tortoisesvn/trunk/doc/issuetrackers.txt].

5.25. TortoiseSVN's Settings

To find out what the different settings are for, just leave your mouse pointer a second on the edit-box/checkbox... and a helpful tooltip will popup.

5.25.1. General Settings

![Figure 5.30. The Settings Dialog, General Page](image)

This dialog allows you to specify your preferred language, and the Subversion-specific settings.

Language

Selects your user interface language. What else did you expect?

Automatically check for newer versions every week

If checked, TortoiseSVN will contact its download site once a week to see if there is a newer version of the program available.

Exclude/Ignore pattern
Exclude patterns are used to prevent unversioned files from showing up e.g. in the commit dialog. Files matching the patterns are also ignored by an import. Exclude files or directories by typing in the names or extensions. Patterns are separated by spaces e.g. */bin */obj *.bak *.~?? *.jar *.[Tt]mp. The first two entries refer to directories, the other four to files.

Note that the exclude patterns you specify here will also affect other Subversion clients running on your PC, including the command line client.

Caution

If you use the Subversion configuration file to set a global-ignores pattern, it will override the settings you made here. The Subversion configuration file is accessed using the Edit as described below.

This exclude pattern will affect all your projects. It is not versioned, so it will not affect other users. By contrast you can also use the versioned svn:ignore property to exclude files or directories from version control. Read Section 5.11, “Ignoring Files And Directories” for more information.

Set filedates to the "last commit time"

This option tells TortoiseSVN to set the filedates to the last commit time when doing a check-out or an update. Otherwise TortoiseSVN will use the current date. If you are developing software it is generally best to use the current date because build systems normally look at the datestamps to decide which files need compiling. If you use "last commit time" and revert to an older file revision, your project may not compile as you expect it to.

Edit...

... the Subversion configuration file directly. Some settings cannot be modified directly by TortoiseSVN, and need to be set here instead. For more information about the Subversion config file see the Subversion Manual [http://svnbook.red-bean.com/en/1.1/ch07.html#svn-ch-7-sect-1.3.2]. The section on Automatic Property Setting [http://svnbook.red-bean.com/en/1.1/ch07s02.html#svn-ch-7-sect-2.4] is of particular interest, and that is configured here.

Clear Auth Cache

When you authenticate with a subversion server, the username and password are cached locally so you don’t have to keep entering them. To clear everything from that cache, use this button.

5.25.2. Look and Feel Settings
This page allows you to specify which of the TortoiseSVN context menu entries will show up in the main context menu, and which will appear in the TortoiseSVN submenu. By default most items are checked and appear in the submenu.

If you have a very large number of files in your working copy folders, it can take a long time before the context menu appears when you right click on a folder. This is because Subversion fetches the status for all files when you ask for folder status. To avoid this delay you can check the Don't fetch status for context menu box. Be warned that the context menu for folders will not always be correct, and may include items which should not really be there. For example, you will see TortoiseSVN->Show Log for an Added folder, which will not work because the folder is not yet in the repository.

**5.25.2.1. Icon Overlay Settings**
This page allows you to choose the items for which TortoiseSVN will display icon overlays. Network drives can be very slow, so by default icons are not shown for working copies located on network shares. You can even disable all icon overlays, but where's the fun in that?

By default, overlay icons will appear in all open/save dialogs as well as in Windows Explorer. If you want them to appear only in Windows Explorer, check the Show overlays only in explorer box.

The Exclude Paths are used to tell TortoiseSVN for which paths not to show icon overlays and status columns. This is useful if you have some very big working copies containing only libraries which you won't change at all and therefore don't need the overlays. For example:

- `f:\development\SVN\Subversion` will disable the overlays on only that specific folder. You still can see the overlays on all files and folder inside that folder.

- `f:\development\SVN\Subversion*` will disable the overlays on all files and folders whose path starts with `f:\development\SVN\Subversion`. That means you won't see overlays for any files and folders below that path.

The same applies to the Include Paths. Except that for those paths the overlays are shown even if the overlays are disabled for that specific drive type.

### 5.25.2.2. Icon Set Selection
Figure 5.33. The Settings Dialog, Icon Set Page

You can change the overlay icon set to the one you like best. Note that if you change overlay set, you may have to restart your computer for the changes to take effect.

5.25.2.3. TortoiseSVN Dialog Settings
Figure 5.34. The Settings Dialog, Dialogs Page

This dialog allows you to configure some of TortoiseSVN's dialogs the way you like them.

Default number of log messages
Limits the number of log messages that TortoiseSVN fetches when you first select TortoiseSVN->Show Log Useful for slow server connections. You can always use Get All or Next 100 to get more messages.

Font for log messages
Selects the font face and size used to display the log message itself in the middle pane of the Revision Log dialog, and when composing log messages in the Commit dialog.

Short date / time format in log messages
If the standard long messages use up too much space on your screen use the short format.

Use old APIs to fetch log messages
As of version 1.2, Subversion servers support fetching a defined number of log messages, rather than the log messages from a range of revisions. With the new API, if you ask for 100 messages you will get 100 message. With the old API, if you ask for 100 messages you get the messages from the last 100 repository revisions. If your file didn't change often, you will see few messages.

However, if you query a pre-1.2 server using the new API, it will return all log message which the client then filters. This can result in very slow responses. If you have to work with an older server, you should check this box to avoid the slowdown. The best option of course is to upgrade the server.

Progress Dialog
TortoiseSVN can automatically close all progress dialogs when the action is finished without error. This setting allows you to select the conditions for closing the dialogs. The default (recommended) setting is Close manually which allows you to review all messages and check what has happened. However, you may decide that you want to ignore some types of message and have the dialog close automatically if there are no critical changes.

Auto-close if no merges, adds or deletes means that the progress dialog will close if there were simple updates, but if changes from the repository were merged with yours, or if any files were added or deleted, the dialog will remain open.

Auto-close if no conflicts relaxes the criteria further and will close the dialog even if there were merges, adds or deletes. However, if there were any conflicts or errors, the dialog remains open.

Auto-close if no errors always closes the dialog even if there were conflicts. The only condition that keeps the dialog open is an error condition, which occurs when Subversion is unable to complete the task. For example, an update fails because the server is inaccessible, or a commit fails because the working copy is out-of-date.

Use autocompletion of filepaths and keywords
The commit dialog includes a facility to parse the list of filenames being committed. When you type the first 3 letters of an item in the list, the autocompletion box pops up, and you can press Enter to complete the filename. Check the box to enable this feature.

5.25.3. Network Settings
Figure 5.35. The Settings Dialog, Network Page

Here you can configure your proxy server, if you need one to get through your company's firewall.

You can also specify which program TortoiseSVN should use to establish a secure connection to a svn+ssh repository. We recommend that you use TortoisePlink.exe. This is a version of the popular Plink program, and is included with TortoiseSVN.

5.25.4. External Program Settings
Figure 5.36. The Settings Dialog, Diff Viewer Page

Here you can define your own diff/merge programs that TortoiseSVN should use. The basic setting is to use the builtin TortoiseMerge which is installed alongside TortoiseSVN.

5.25.4.1. Diff Viewer

An external diff program may be used for comparing different revisions of files. The external program will need to obtain the filenames from the command line, along with any other command line options. TortoiseSVN uses substitution parameters prefixed with %, when it encounters one of these it will substitute the appropriate value. The order of the parameters will depend on the Diff program you use.

%base
    The original file without your changes
%bname
    The window title for the base file
%mine
    Your own file, with your changes
%yname
    The window title for your file

The window titles are not pure filenames. TortoiseSVN treats that as a name to display and creates the names accordingly. So e.g. if you're doing a diff from a file in revision 123 with a file in your working copy, the names will be `filename : revision 123` and `filename : working copy`

For example, with ExamDiff Pro:

```
C:\Path-To\ExamDiff.exe %base %mine
```
or with KDiff3:

```
C:\Path-To\kdiff3.exe %base %mine --L1 %bname --L2 %yname
```

If you use the `svn:keywords` property to expand keywords, and in particular the `revision` of a file, then there may be a difference between files which is purely due to the current value of the keyword. TSVN can hide this artificial difference by first parsing the files for keywords before doing the diff operation. However, this can take some time, especially with large files, so this behaviour is optional. If Convert files when diffing against BASE is checked then TSVN will pre-process the files.

### 5.25.4.2. Merge Tool

An external merge program used to resolve conflicted files. Parameter substitution is used in the same way as with the Diff Program.

- `%base`: the original file without your or the others changes
- `%bname`: The window title for the base file
- `%mine`: your own file, with your changes
- `%yname`: The window title for your file
- `%theirs`: the file as it is in the repository
- `%tname`: The window title for the file in the repository
- `%merged`: the conflicted file, the result of the merge operation
- `%mname`: The window title for the merged file

For example, with Perforce WinMerge:

```
C:\Path-To\P4WinMrg.exe %base %theirs %mine %merged
```

or with KDiff3:

```
C:\Path-To\kdiff3.exe %base %mine %theirs --o %merged
```

### 5.25.4.3. Unified Diff Viewer

A viewer program for unified-diff files (patch files). If you don't have one the builtin option is to use NotePad. No parameters are required.

The original Windows NotePad program does not behave well on files which do not have standard CR-LF line-endings. Since most unified diff files have pure LF line-endings, they do not view well in NotePad. However, you can download a free NotePad replacement Notepad2
[http://www.flos-freeware.ch/notepad2.html] which not only displays the line-endings correctly, but also colour codes the added and removed lines.

![Advanced Diff Settings](image)

**Figure 5.37. The Settings Dialog, Diff/Merge Advanced Dialog**

In the advanced settings, you can define a different diff and merge program for every file extension. For instance you could associate Photoshop as the “Diff” Program for .jpg files :-)

### 5.25.5. TortoiseSVN’s Sounds

TortoiseSVN has three custom sounds which are installed by default.

- Error
- Notice
- Warning

You can select different sounds (or turn these sounds off completely) using the Windows Control Panel, not the TortoiseSVN settings dialog.

### 5.26. Final Step

**Wishlist**

Even though TortoiseSVN and TortoiseMerge are free, you can support the developers by sending in patches and play an active role in the development. You can also help to cheer us up during the endless hours we spend in front of our computers.
While working on TortoiseSVN we love to listen to music. And since we spend many hours on the project we need a lot of music. Therefore we have set up some wish-lists with our favourite music CD's and DVD's: http://tortoisesvn.tigris.org/donate.html Please also have a look at the list of people who contributed to the project by sending in patches or translations.
Appendix A. Frequently Asked Questions (FAQ)

Because TortoiseSVN is being developed all the time it is sometimes hard to keep the documentation completely up to date. We maintain an interactive online FAQ at Berlios.de [http://tortoisesvn.berlios.de/faq/] which contains a selection of the questions we are asked the most on the TortoiseSVN mailing list <dev@tortoisesvn.tigris.org>.

We also maintain a project Issue Tracker at Berlios.de [http://tortoisesvn.berlios.de/issues/index.php?order=id&sort=desc&status=all] which tells you about some of the things we have on our To-Do list, and bugs which have already been fixed. If you think you have found a bug, or want to request a new feature, check here first to see if someone else got there before you.

If you have a question which is not answered anywhere else, the best place to ask it is on the mailing list.
Appendix B. How Do I...

This appendix contains solutions to problems/questions you might have when using TortoiseSVN.

B.1. Move/copy a lot of files at once

Moving/Copying single files can be done by using TortoiseSVN->Rename.... But if you want to move/copy a lot of files, this way is just too slow and too much work.

The recommended way is by right-dragging the files to the new location. Simply right-click on the files you want to move/copy without releasing the mouse button. Then drag the files to the new location and release the mouse button. A context menu will appear where you can either choose Context Menu->Copy in Subversion to here, or Context Menu->Move in Subversion to here.

B.2. Force users to enter a log message

There are two ways to prevent users from committing with an empty log message. One is specific to TortoiseSVN, the other works for all Subversion clients, but requires access to the server directly.

B.2.1. Hook-script on the server

If you have direct access to the repository server, you can install a pre-commit hook script which rejects all commits with an empty or too short log message.

In the repository folder on the server, there's a subfolder hooks which contains some example hook scripts you can use. The file pre-commit.tmpl contains a sample script which will reject commits if no log message is supplied, or the message is too short. The file also contains comments on how to install/use this script. Just follow the instructions in that file.

This method is the recommended way if your users also use other Subversion clients than TortoiseSVN. The drawback is that the commit is rejected by the server and therefore users will get an error message. The client can't know before the commit that it will be rejected. If you want to make TortoiseSVN have the OK button disabled until the log message is long enough then please use the method described below.

B.2.2. Project properties

TortoiseSVN uses properties to control some of its features. One of those properties is the $tsvn:minlogsize property.

If you set that property on a folder, then TortoiseSVN will disable the OK button in all commit dialogs until the user has entered a log message with at least the length specified in the property.

For detailed information on those project properties, please refer to Section 5.15, “Project Settings”

B.3. Update selected files from the repository

Normally you update your working copy using TortoiseSVN->Update. But if you only want to pick up some new files that a colleague has added without merging in any changes to other files at the same time, you need a different approach.

Use TortoiseSVN->Check for Modifications, and click on Check repository to see what has changed in the repository. Select the files you want to update locally, then use the context menu to update just those files.

B.4. Roll back revisions in the repository

B.4.1. Use the revision log dialog

The easiest way to revert the changes from a single revision is to use the revision log dialog.
1. Select the file or folder in which you need to revert the changes. If you want to revert all changes, this should be the top level folder.

2. Select TortoiseSVN->Show Log to display a list of revisions. You may need to use Get All or Next 100 to show the revision you are interested in.

3. right click on the revision you wish to undo, then select Context Menu->Revert changes from this revision.

You have reverted the changes within your working copy. Check the results, then commit the changes.

B.4.2. Use the merge dialog

To undo a larger range of revisions, you can use the Merge dialog. The previous method uses merging behind the scenes; this method uses it explicitly.

1. In our working copy select TortoiseSVN->Merge.

2. In the From: field enter the full folder url of the branch or tag containing the changes you want to revert in your working copy. This should come up as the default URL.

3. In the From Revision field enter the revision number that you are currently at. If you are sure there is no-one else making changes, you can use the HEAD revision.

4. make sure the Use “From:” URL checkbox is checked.

5. In the To Revision field enter the revision number that you want to revert to, ie. the one before the first revision to be reverted.

6. Click OK to complete the merge.

You have reverted the changes within your working copy. Check the results, then commit the changes.

B.4.3. Use svndumpfilter

Since TortoiseSVN never loses data, your “rolled back” revisions still exist as intermediate revisions in the repository. Only the HEAD revision was changed to a previous state. If you want to make revisions disappear completely from your repository, erasing all trace that they ever existed, you have to use more extreme measures. Unless there is a really good reason to do this, it is not recommended. One possible reason would be that someone committed a confidential document to a public repository.

The only way to remove data from the repository is to use the Subversion command line tool svnadmin. You can find a description of how this works in the Subversion Book [http://svnbook.red-bean.com/en/1.1/ch05s03.html#svn-ch-5-sect-3.1.3].

B.5. Compare two revisions of a file

If you want to compare two revisions in a file's history, for example revisions 100 and 200 of the same file, just use TortoiseSVN->Show Log to list the revision history for that file. Pick the two revisions you want to compare then use Context Menu->Compare Revisions.

If you want to compare the same file in two different trees, for example the trunk and a branch, you can use the repository browser to open up both trees, select the file in both places, then use Context Menu->Compare Revisions.

If you want to compare two trees to see what has changed, for example the trunk and a tagged release, you can use TortoiseSVN->Revision Graph Select the two nodes to compare, then use
Context Menu->Compare HEAD Revisions. This will show a list of changed files, and you can then select individual files to view the changes in detail. Alternatively use Context Menu->Unified Diff of HEAD Revisions to see a summary of all differences, with minimal context.

B.6. Include a common sub-project

Sometimes you will want to include another project within your working copy, perhaps some library code. You don't want to make a duplicate of this code in your repository because then you would lose connection with the original (and maintained) code. Or maybe you have several projects which share core code. There are at least 3 ways of dealing with this.

B.6.1. Use svn:externals

Set the `svn:externals` property for a folder in your project. The property value should be the name of a subfolder which you want the common code checked into, and the repository URL that you want to be checked out there. You can add multiple folder/URL pairs to one folder. For example:

```
third-party/sounds http://sounds.red-bean.com/repos
third-party/skins http://skins.red-bean.com/repositories/skinproj
```

Of course this code has to reside in a Subversion repository, although it does not have to be the same one.

Commit the new folder. Now when you update, Subversion will pull a copy of that project from its repository into your working copy. The subfolders will be created automatically if required. Each time you update your main working copy, you will also receive the latest version of all external projects.

If the external project is in the same repository, any changes you make there will be included in the commit list when you commit your main project.

If the external project is in a different repository, any changes you make to the external project will be notified when you commit the main project, but you have to commit those external changes separately.

B.6.2. Use a nested working copy

Create a new folder within your project to contain the common code, but do not add it to Subversion.

Select TortoiseSVN->Checkout for the new folder and checkout a copy of the common code into it. You now have a separate working copy nested within your main working copy.

The two working copies are independent. When you commit changes to the parent, changes to the nested WC are ignored. Likewise when you update the parent, the nested WC is not updated.

B.6.3. Use a relative location

If you use the same common core code in several projects, and you do not want to keep multiple working copies of it for every project that uses it, you can just check it out to a separate location which is related to all the other projects which use it. For example:

```
C:\Projects\Proj1
C:\Projects\Proj2
C:\Projects\Proj3
C:\Projects\Common
```

and refer to the common code using a relative path, eg. `../..\Common\DSPcore`.

If your projects are scattered in unrelated locations you can use a variant of this, which is to put the common code in one location and use drive letter substitution to map that location to something you can hard code in your projects, eg. Checkout the common code to `D:\Documents\Framework`
or C:\Documents and Settings\(login)\My Documents\framework then use

SUBST X: "D:\Documents\framework"

to create the drive mapping used in your source code. Your code can then use absolute locations.

#include "X:\superio\superio.h"

This method will only work in an all-PC environment, and you will need to document the required drive mappings so your team know where these mysterious files are. This method is strictly for use in closed development environments, and not recommended for general use.

**B.7. Create a shortcut to a repository**

If you frequently need to open the repository browser at a particular location, you can create a desktop shortcut using the automation interface to TortoiseProc. Just create a new shortcut and set the target to:

TortoiseProc.exe /command:repobrowser /path:"url/to/repository" /notempfile

Of course you need to include the real repository URL.
Appendix C. Automating TortoiseSVN

Since all commands for TortoiseSVN are controlled through command line parameters, you can automate it with batch scripts or start specific commands and dialogs from other programs (e.g. your favourite text editor).

For most use cases you might want to use the official Subversion command line client to do automation. But if you want GUI dialogs for still required user input or progress information you can use TortoiseSVN.

C.1. TortoiseSVN Commands

All commands are specified with the parameter /command:abcd where abcd is the required command name. Most of these commands need at least one path argument, which is given with /path:"some\path". In the following table the command refers to the /command:abcd parameter and the path refers to the /path:"some\path" parameter.

Since some of the commands can take a list of target paths (e.g. committing several specific files) the /path parameter can take several paths, separated by a * character.

Since TortoiseSVN uses temporary files to pass multiple arguments between the shell extension and the main program, you must add the /notempfile parameter! If you don't, the command won't work and the file you pass with the /path parameter will be deleted!

The progress dialog which is used for commits, updates and many more commands usually stays open after the command has finished until the user presses the OK button. This can be changed by checking the corresponding option in the settings dialog. But using that setting will close the progress dialog, no matter if you start the command from your batchfile or from the TortoiseSVN context menu.

To close the progress dialog at the end of a command automatically without using the permanent setting you can pass the /closeonend parameter.

- /closeonend:0 don't close the dialog automatically
- /closeonend:1 auto close if no errors
- /closeonend:2 auto close if no conflicts
- /closeonend:3 auto close if no merges

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>about</td>
<td>Shows the About-dialog. This is also shown if no command is given.</td>
</tr>
<tr>
<td>log</td>
<td>Opens the log dialog. The path specifies the file or folder for which the log should be shown. Three additional options can be set: /revstart:xxx, /revend:xxx and /strict</td>
</tr>
<tr>
<td>checkout</td>
<td>Opens the checkout dialog. The /path specifies the target directory and the /url specifies the URL to checkout from.</td>
</tr>
<tr>
<td>import</td>
<td>Opens the import dialog. The path specifies the directory with the data to import.</td>
</tr>
<tr>
<td>update</td>
<td>Updates the working copy in /path to HEAD. If the option /rev is given then a dialog is shown to ask the user to which revision the update should go.</td>
</tr>
<tr>
<td>commit</td>
<td>Opens the commit dialog. The path specifies the target directory or the list of files to commit. You can also specify the /logmsg switch to pass a predefined log message to the commit dialog.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>add</td>
<td>Adds the files in /path to version control.</td>
</tr>
<tr>
<td>revert</td>
<td>Reverts local modifications of a working copy. The /path tells which items to revert.</td>
</tr>
<tr>
<td>cleanup</td>
<td>Cleans up interrupted or aborted operations and unlocks the working copy in /path.</td>
</tr>
<tr>
<td>resolve</td>
<td>Marks a conflicted file specified in /path as resolved. If /noquestion is given, then resolving is done without asking the user first if it really should be done.</td>
</tr>
<tr>
<td>repocreate</td>
<td>Creates a repository in /path</td>
</tr>
<tr>
<td>switch</td>
<td>Opens the switch dialog. The path specifies the target directory.</td>
</tr>
<tr>
<td>export</td>
<td>Exports the working copy in /path to another directory. If the /path points to an unversioned directory, a dialog will ask for an URL to export to the dir in /path.</td>
</tr>
<tr>
<td>merge</td>
<td>Opens the merge dialog. The path specifies the target directory.</td>
</tr>
<tr>
<td>copy</td>
<td>Brings up the branch/tag dialog. The /path is the working copy to branch/tag from.</td>
</tr>
<tr>
<td>settings</td>
<td>Opens the settings dialog.</td>
</tr>
<tr>
<td>remove</td>
<td>Removes the file(s) in /path from version control.</td>
</tr>
<tr>
<td>rename</td>
<td>Renames the file in /path. The new name for the file is asked with a dialog.</td>
</tr>
<tr>
<td>diff</td>
<td>Starts the external diff program specified in the TortoiseSVN settings. The /path specifies the first file. If the option /path2 is set, then the diff program is started with those two files. If /path2 is omitted, then the diff is done between the file in /path and its BASE.</td>
</tr>
<tr>
<td>conflicteditor</td>
<td>Starts the conflicteditor specified in the TortoiseSVN settings with the correct files for the conflicted file in /path.</td>
</tr>
<tr>
<td>relocate</td>
<td>Opens the relocate dialog. The /path specifies the working copy path to relocate.</td>
</tr>
<tr>
<td>help</td>
<td>Opens the help file.</td>
</tr>
<tr>
<td>repostatus</td>
<td>Opens the check-for-modifications dialog. The path specifies the working copy directory.</td>
</tr>
<tr>
<td>repobrowser</td>
<td>Starts the repository browser dialog, pointing to the URL of the working copy given in /path or /path points directly to an URL. An additional option /rev:xxx can be used to specify the revision which the repository browser should show. If the /rev:xxx is omitted, it defaults to HEAD.</td>
</tr>
<tr>
<td>ignore</td>
<td>Adds all targets in /path to the ignore list, i.e. adds the svn:ignored property to those files.</td>
</tr>
<tr>
<td>blame</td>
<td>Opens the blame dialog for the file specified in /path.</td>
</tr>
<tr>
<td>cat</td>
<td>Saves a file from an URL or working copy path given in /path to the location given in /savepath:path. The revision is given in /revision:xxx. This can be used to get a file with a specific revision.</td>
</tr>
<tr>
<td>createpatch</td>
<td>Creates a patch file for the path given in /path.</td>
</tr>
<tr>
<td>revisiongraph</td>
<td>Shows the revision graph for the path given in /path.</td>
</tr>
<tr>
<td>lock</td>
<td>Locks a file. The 'lock' dialog is shown so the user can enter a comment for the lock. /path</td>
</tr>
<tr>
<td>unlock</td>
<td>Unlocks a file. /path</td>
</tr>
</tbody>
</table>

Table C.1. List of available command line options
Examples (which should be entered on one line):

TortoiseProc.exe /command:commit /path:"c:\svn_wc\file1.txt*c:\svn_wc\file2.txt" /logmsg:"test log message" /notempfile /closeonend

TortoiseProc.exe /command:update /path:"c:\svn_wc\" /notempfile /closeonend

TortoiseProc.exe /command:log /path:"c:\svn_wc\file1.txt" /revstart:50 /revend:60 /notempfile /closeonend
Index

A
Access, 27
add, 51
annotate, 70
Apache, 14
authentication, 31
automation, 93

B
backup, 27
blame, 70
branch, 59
bugtracker, 75
bugtracking, 75

C
checkout, 35
cleanup, 56
command line, 93
commit, 36
common projects, 91
compare files, 90
conflict, 8, 40
copy, 59
Create
   Command Line Client, 26
   TortoiseSVN, 26

D
delete, 53
diff, 50
diff tools, 50
diffing, 44
domaincontroller, 19
drag handler, 30
drag-n-drop, 30

E
edit log/author, 49
exclude pattern, 34, 77
explorer, 1
Explorer Columns, 43
export, 36
external repositories, 35
externals, 35

F
filter, 49

G
global ignore, 77
graph, 72

H
history, 45
hooks, 28

I
ignore, 52
import, 32
Index of projects, 18
install, 2
issuetracker, 75

L
locking, 65
log, 45
log message, 89

M
merge, 62
merge tools, 50
mod_authz_svn, 15
move, 53
moving, 89

O
overlays, 41

P
patch, 69
praise, 70
properties, 56
proxy server, 82

R
relocate, 74
remove, 53
rename, 53
reopbrowser, 71
repository, 4
Repository, 32
revert, 55
revision, 11, 72
rollback, 89

S
settings, 77
shortcut, 92
sounds, 86
special files, 34
spellchecker, 3
SSL, 21
SSPI, 19
statistics, 50
status, 41, 43
Subversion book, 4
SVNParentPath, 16, 17
SVNPath, 16
svnservce, 23
switch, 61

96
T
tag, 59
  temporary files, 33

U
UNC paths, 27
undo, 55
update, 39, 89

V
version control, 1

W
WebDAV, 14
Windows domain, 19
Windows shell, 1
working copy, 9