Subversion How-To

Last Revised: 2008-02-22

Table of Contents

Authors....1
Legal....1
Introduction....1
Checkout with Tortoise....2
Adding and Deleting Files....3
Update Files to the Newest Revision....5
Commit Changes....5
Reverting to a Current Version....8
Moving Files and Folders....8
Conflict Resolution....10
Browsing the Log....12
Updating To Older Revision....13
Etiquette....14
Command Line Client....17
References:....17

Authors

The following people have contributed to this document.

- Nick Guertin (BS.CPE 2008)
- David Ransom (BS.CPE 2009)
- Peter C. Chapin (ECET Faculty)

Legal

Introduction

Subversion is a *version control system*. It is useful when several people work on a collection of shared files. Subversion tracks all changes made to any file and allows you to access previous versions of files. It also makes it easy to share the latest updates of a file with others. To use Subversion you first need to

have a Subversion client installed. We recommend TortoiseSVN for Windows and most of this document describes how to use that client. However, a later section of this document briefly describes how to use the command line client typically found on Unix-like systems. Once the client is installed you can then check out some or all of the files stored in a Subversion *repository* (on a Subversion server). Your copy of these files is stored in an ordinary folder called a *working copy*.

You can edit the files in your working copy as you see fit (coordinating with the other people who might also be working on their own copy of those files) and then, when you are ready, commit your changes back to the repository. Other team members can pick up your changes by updating their working copies. In this way everyone keeps their working copies synchronized with each other and with the central repository.

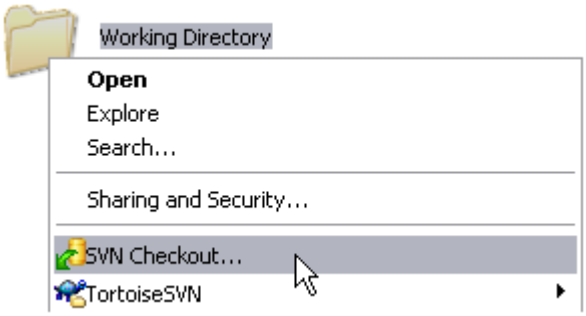
If a change is made that is later discovered to be bad, that change can be reversed by making use of the historical information stored by the Subversion server. In addition the server logs who made each change, when each change was made, and what files were involved in each change. This makes tracking down problems much easier than it would be the case without this information.

This document is a tutorial on how to use Subversion. It is organized by task, in an order that mimics the order in which you might want to perform those tasks. The more advanced material is at the end; feel free to skip whatever parts you want.

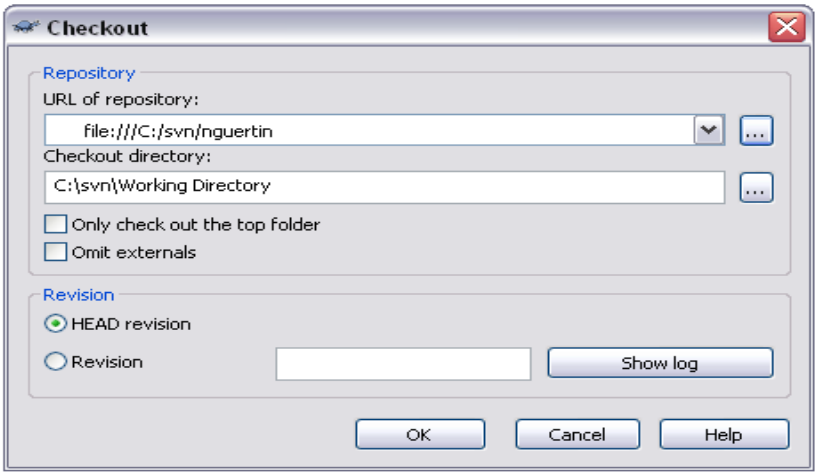
This document assumes you are comfortable with using Windows, but you do not need to be a Windows expert. In addition, this document assumes you've never used a version control system before.

Checkout with Tortoise

1. Create a new folder where you want to store your working copy.
2. Right click on this folder and select “SVN Checkout...”



3. A new window should appear similar to the one shown below.

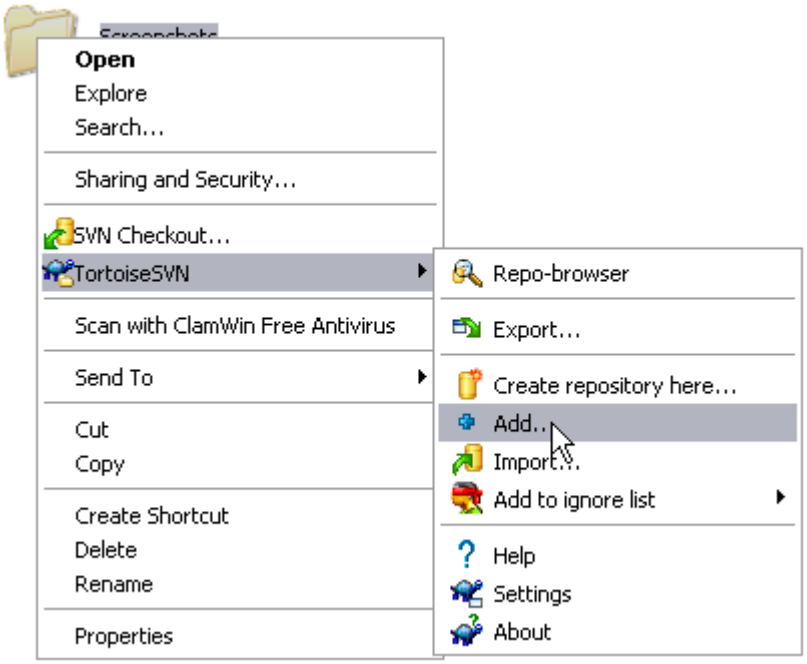


4. In the first text box enter the location of the repository where the files you want are stored, such as svn://svn.ecet.vtc.edu/Scratch/trunk. If you are not sure what URL to use for your project, you should ask someone who is working on the project or the repository administrator.
5. In the second text box, the location of the folder that you selected should be there, if it is not then either use the "..." button on the right to browse for it or enter it in manually.
6. For now just leave the revision set to HEAD and don't worry about anything else as it will be covered later. HEAD just means that the most recent version will be checked out.
7. Now select OK. This will bring you to a screen where it will show you all the files and folders that are added to your working directory. Select OK.
8. Now your folder is populated with the latest revision of the project and will have a green circle icon with a check mark inside it to signify this. Whenever you make changes this icon will turn into a red circle with an exclamation point in it.

9. You have now successfully checked out a group of version controlled files. Good job!

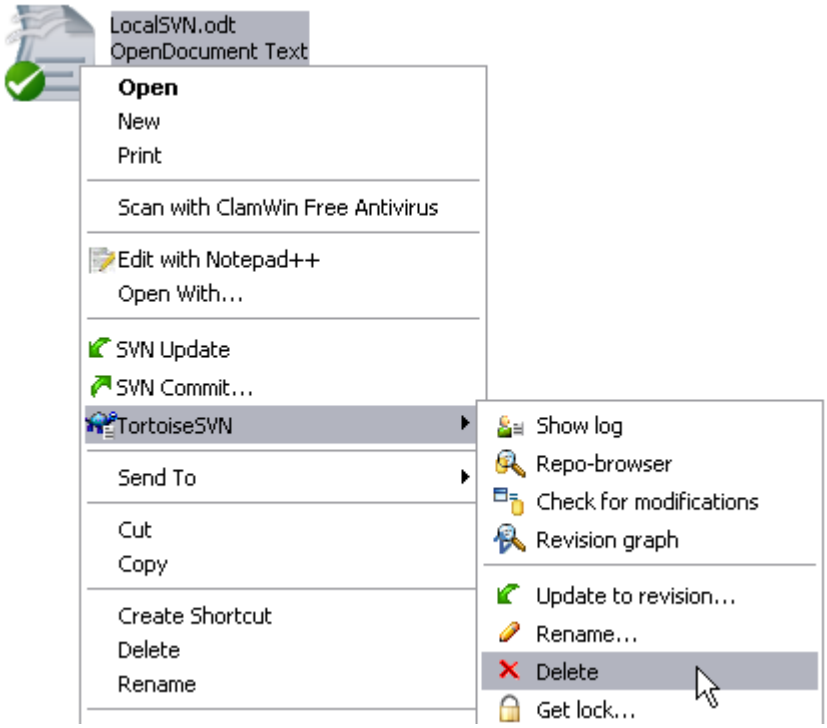
Adding and Deleting Files

1. First we'll look at how to add files to the repository. Browse to a version controlled directory and either move a file you want controlled to that directory or select a file that's currently unversioned. Right-click and select TortoiseSVN -> Add as shown below.



2. You will be prompted with a box listing the file you are trying to add. If you are adding a folder, all the files in that folder will be shown as well and you should unselect any files that you don't want in the repository. When you click on OK, the file or files should now have a blue "+" on them. These will be added at your next commit.

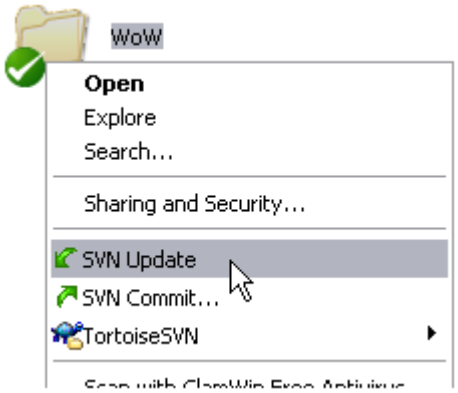
3. Deleting files from the repository works pretty much the same way. Don't even think about taking the "easy" route and just hitting the delete key on the files you don't want anymore, next time you update they'll be right back as the changes didn't make it to the repository. Instead, browse to the file you want removed from the repository, right-click and select TortoiseSVN -> Delete as shown below.



4. Once you click delete the file will disappear. On your next commit the file will no longer be under version control and will not be included in future updates. Checking out older copies before the deletion will still include the file. If you deleted by mistake, move to the parent folder and revert.

Update Files to the Newest Revision

1. Using Windows Explorer, browse to your working copy.
2. Right-click on the folder and select SVN Update as shown below. All of your files should now be at the newest version. If you have made changes to any file your changes will not be erased and only updates elsewhere will be applied.

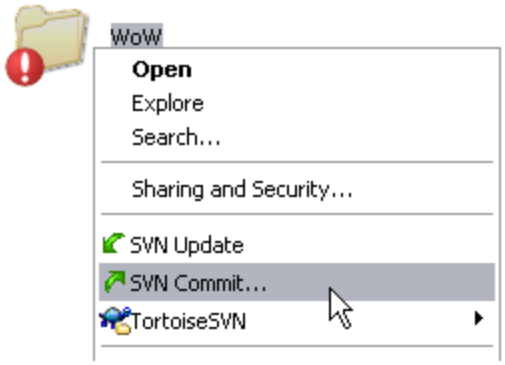


Commit Changes

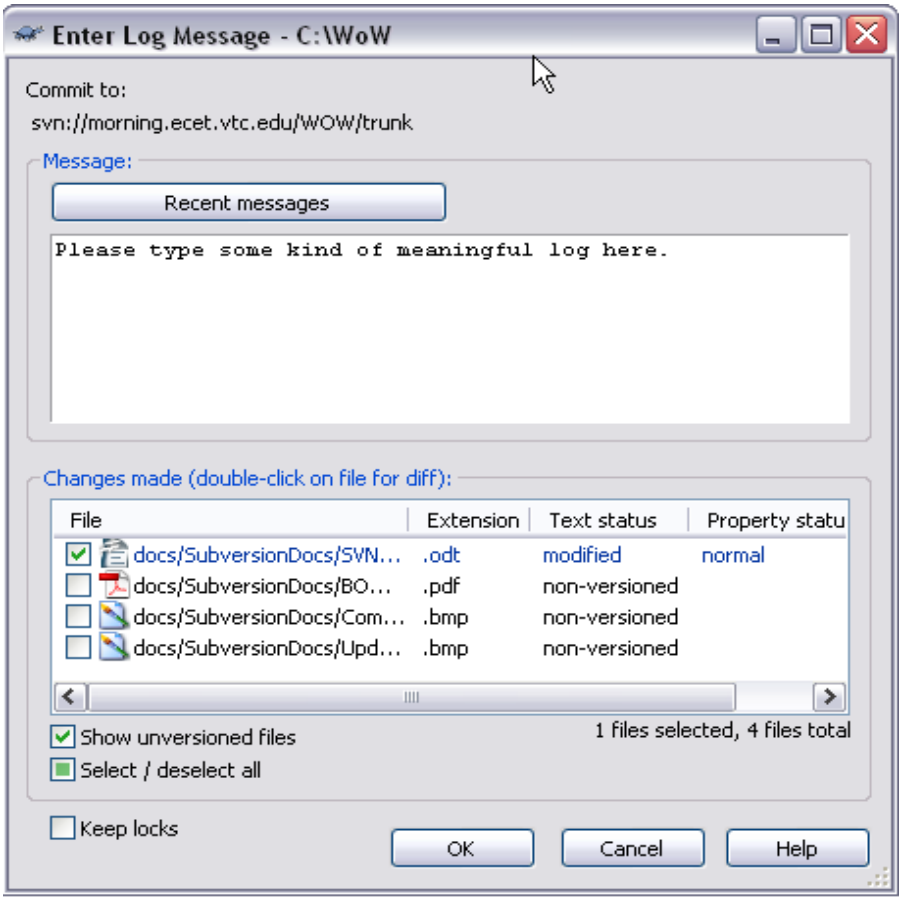
1. After you have made some changes to files in your working copy and everything works as it should, you're going to want to save them in the repository. You can only commit when your

working copy has changed to a red circle with a "!".

2. First you should update your working directory as described above. This will not do anything with the changes you have made, but will update everything else so that you are working with the newest revision.
3. If you are working on a program and the update does update a few files you may want to try building again to make sure that the new changes along with yours haven't broken something. Better to take an extra minute than to commit something that breaks the program for others.
4. Now that you're at the newest revision and have checked that everything still works, just right-click on your local copy and select SVN commit.



5. You will now be asked to enter a log message before you commit, as shown below. It is very important that you write something meaningful for a log message otherwise it will be hard to find a specific version where something was broken or a certain change was made. A good message typically contains the changes you made (not just the file you changed) and why you made those changes (bug fix, new feature, updating documentation, etc.). Note that you do not need to mention your name or the date in the log message. The Subversion server records these things automatically.

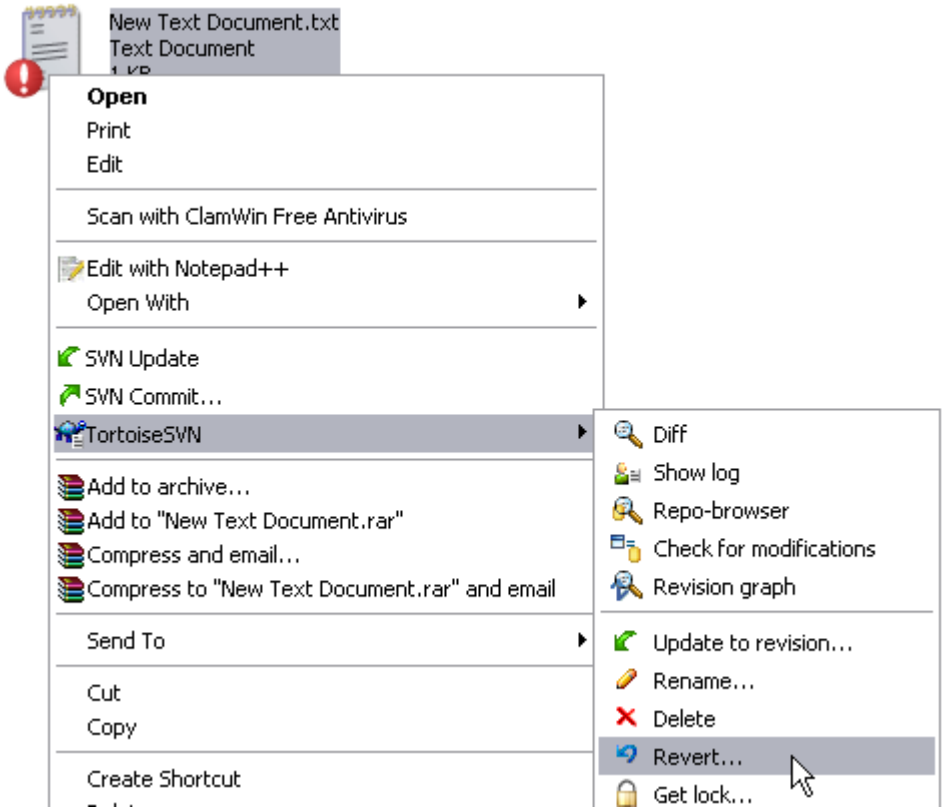


6. In the changes made window you can select the changes you want submitted and which you don't. For now don't worry about this, subversion will select the changed files that are already under version control, so you can just let it work its magic.
7. Hit the OK button to commit. You may be prompted for a user name and password which your teacher or administrator should have given you, just enter it and hit OK.

Reverting to a Current Version

1. Sometimes you will be working in a file, and you decide that you don't want to keep the changes you have made. Subversion will let you revert any changes you have made to your current working version. In order to do this you must have made changes so that there is a red circle with an exclamation point in it on your file. Just right-click on your file, browse to TortoiseSVN->Revert, as shown in the figure below.

2. This will bring you to a dialog screen where it will show you a list of modified files that will be

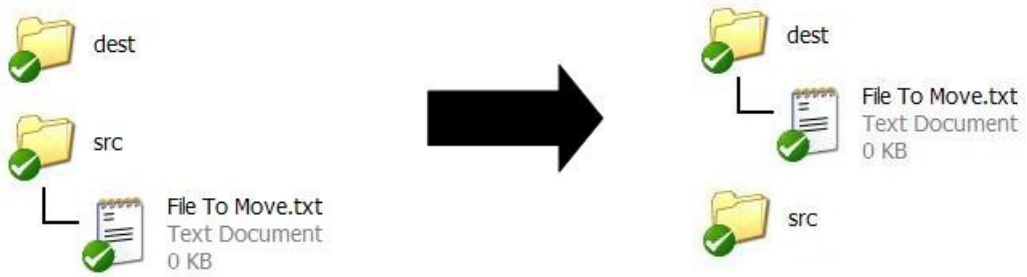


reverted.

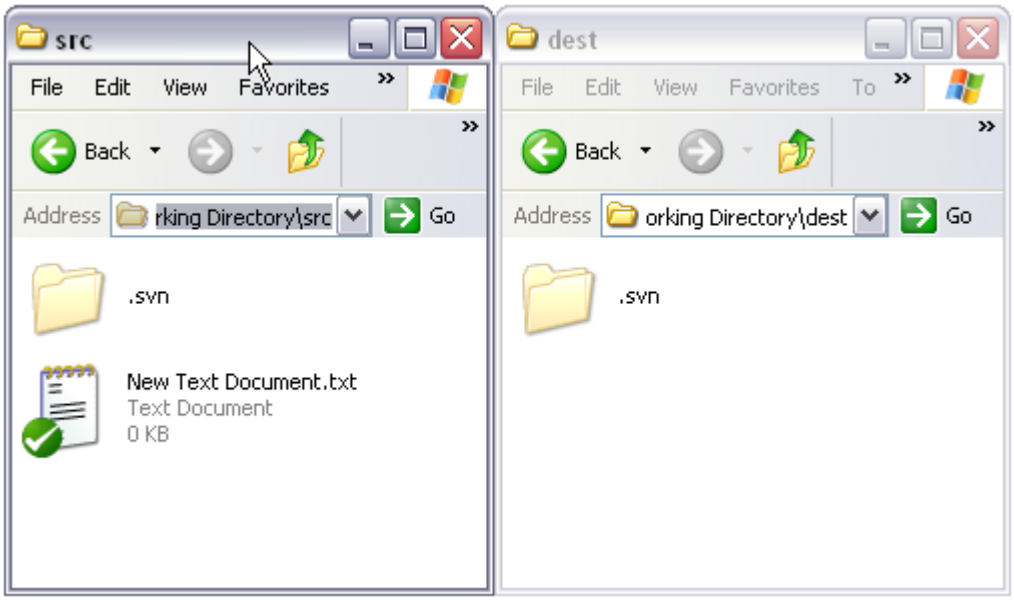
3. Select all the files you wish to revert. Note: This revert will NOT update your copy of the file to the HEAD revision in the central repository. This will only revert back to whatever the most recent version that you had previously checked out on your system.

Moving Files and Folders

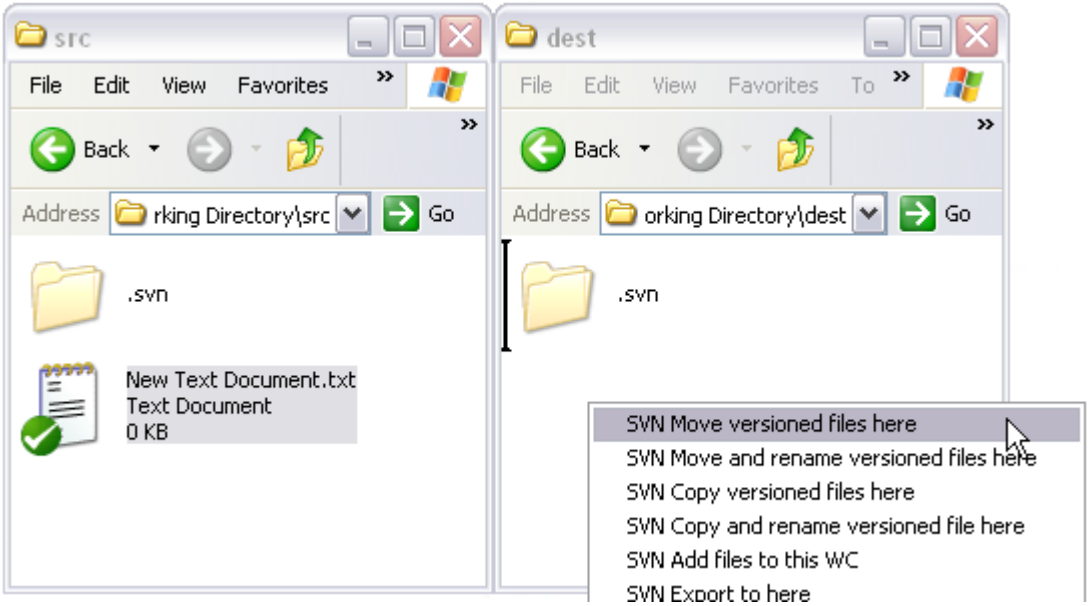
1. If you ever have the need to refactor the directory structure of your repository, you can move files and folders around and commit them back to the central server. But only in a very special way. First, create a destination directory where you want to put your files and add it to the repository.



2. Open Up both of the folders that you want to transfer files. Right-click the file that you want to move, hold it, and drag it from one folder to the next. Release the button and drop your file into the destination folder.



3. When you drop it, a menu will pop up with different options for what you want to do with that file. Select "SVN Move versioned files here".



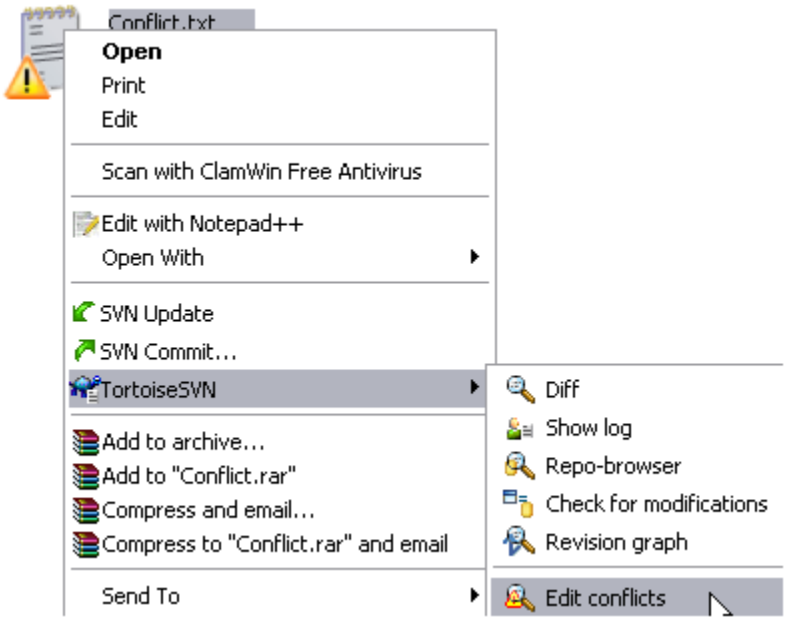
4. This will copy the file, add it to the repository in the new location, and delete the old version all in one step. After that, all you have to do is commit the changes to the repository, and you will have successfully moved a file from one place to another. This technique also works for moving folders.

Conflict Resolution

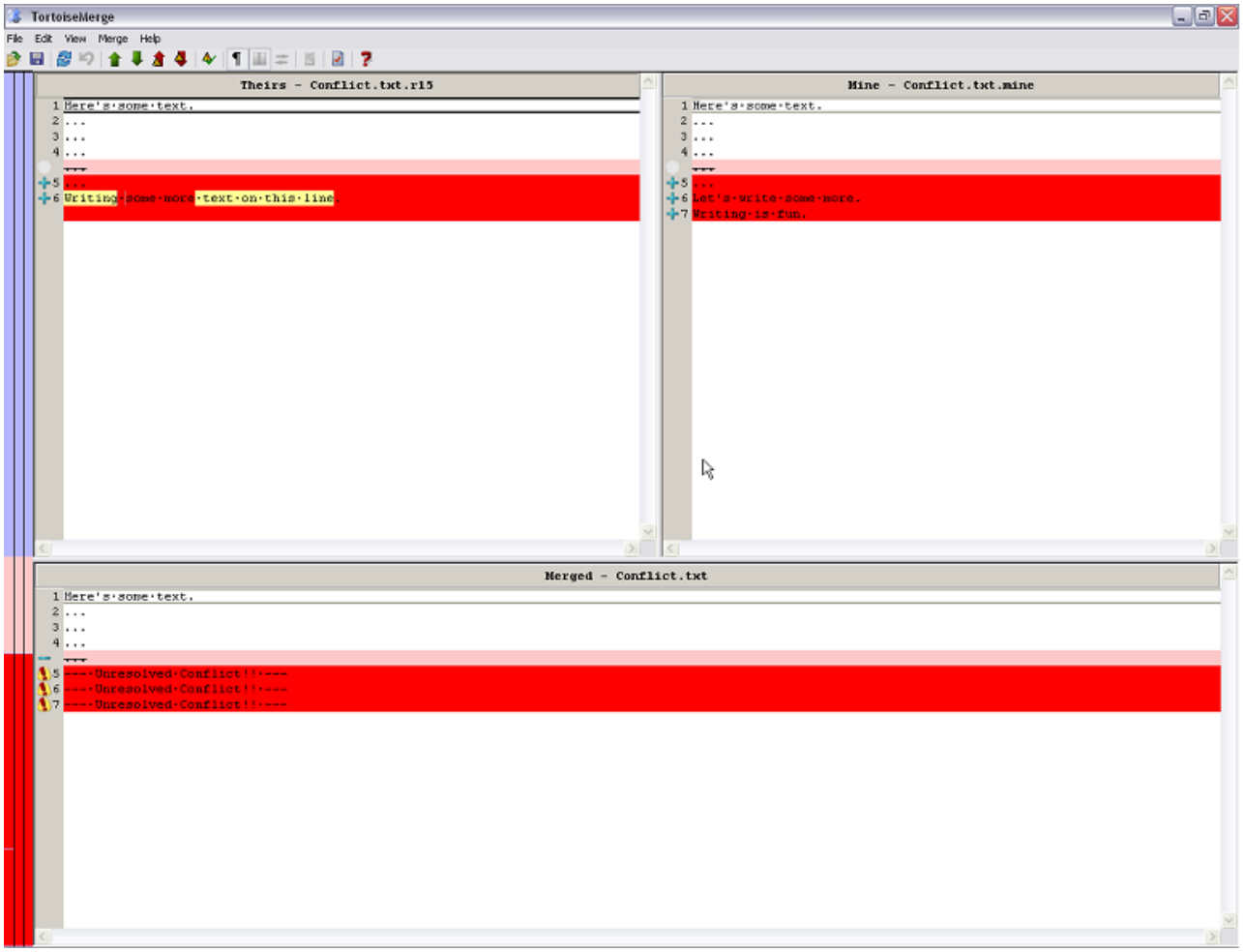
1. Conflicts occur when you and another person modify the same line or lines of a file at about the same time. When one of you commits, the other will have to update before they can commit. But, the same lines were edited in the same file and subversion doesn't know anything about what's being written, so it enters into a conflict state and it's your job to determine which changes are the right ones. When this happens the window showing you the update progress will tell you there's a conflict and conflicted files and folders will receive exclamation points in yellow triangles.

2. Subversion adds a marker of "<<<<<<<" to conflicting lines in your copy to help with the resolution. It also adds 3 new files to the directory, a clean copy of your file labeled with a .mine, a copy of the last revision before it was edited, and a copy of the newest revision; the latter two labeled as .rx, where x is the version number.

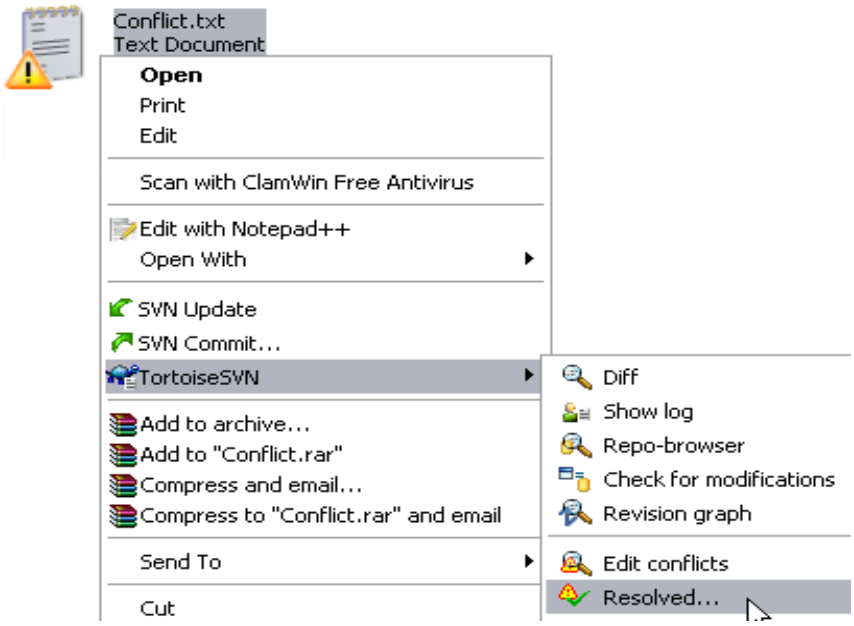
3. To resolve the conflict you can either edit the file manually using whatever tool you've been using, or you can use the conflict editor. We're going to use the conflict editor, so right-click on the file and select TortoiseSVN->Edit Conflicts.



4. In this editor you will see three sets of text: the newest version on the left, your version on the right, and the version resulting from a merge on the bottom. In all three there are red areas which delineate the conflicted portions. Clicking on a line in either of the top two files will select the line, right-clicking will bring up a menu allowing you to choose what to use. The first two selections are fairly self explanatory, the other two selections work by placing the text from one file then placing the text from the other file below it.

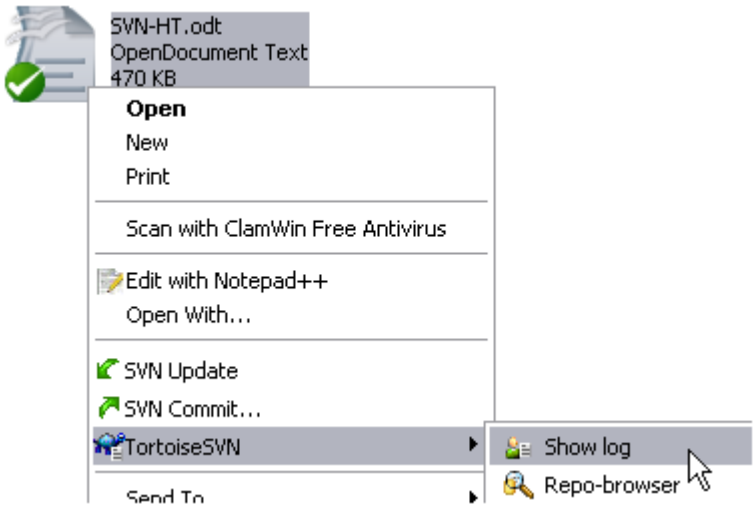


5. Once you have resolved the conflict to your satisfaction, right-click on the file and select TortoiseSVN->Resolved. This will remove the extra files and use your selected fix when you next commit.

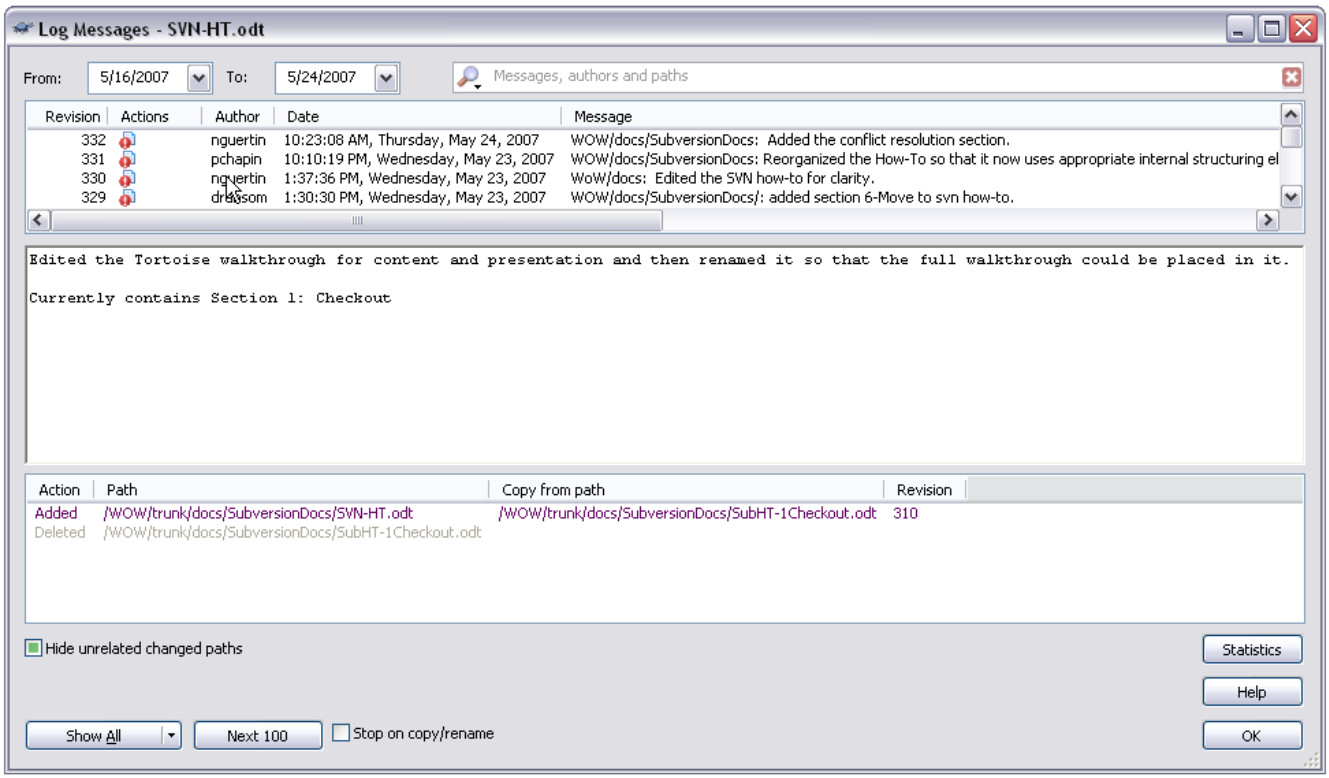


Browsing the Log

1. Browsing log files is extremely useful, allowing you to see past changes, yet it's also very easy to do. Just right-click on a file or folder and click TortoiseSVN->Show Log.

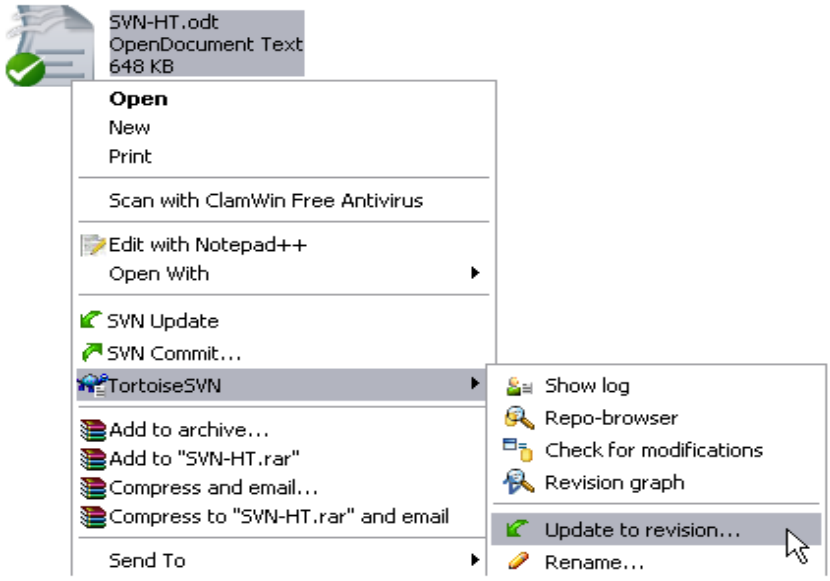


2. This will bring up a window showing the changes to that file. If you selected a folder, you will see all the changes to any and all controlled files in it. Clicking on items in the top section will cause the full log message to be shown in the center and a list of files changed at that time.

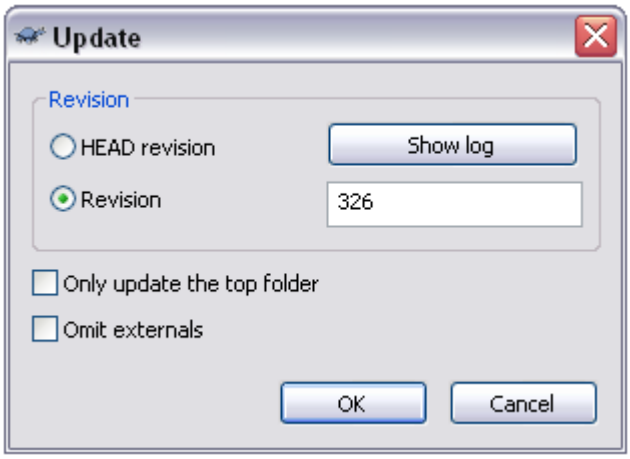


Updating To Older Revision

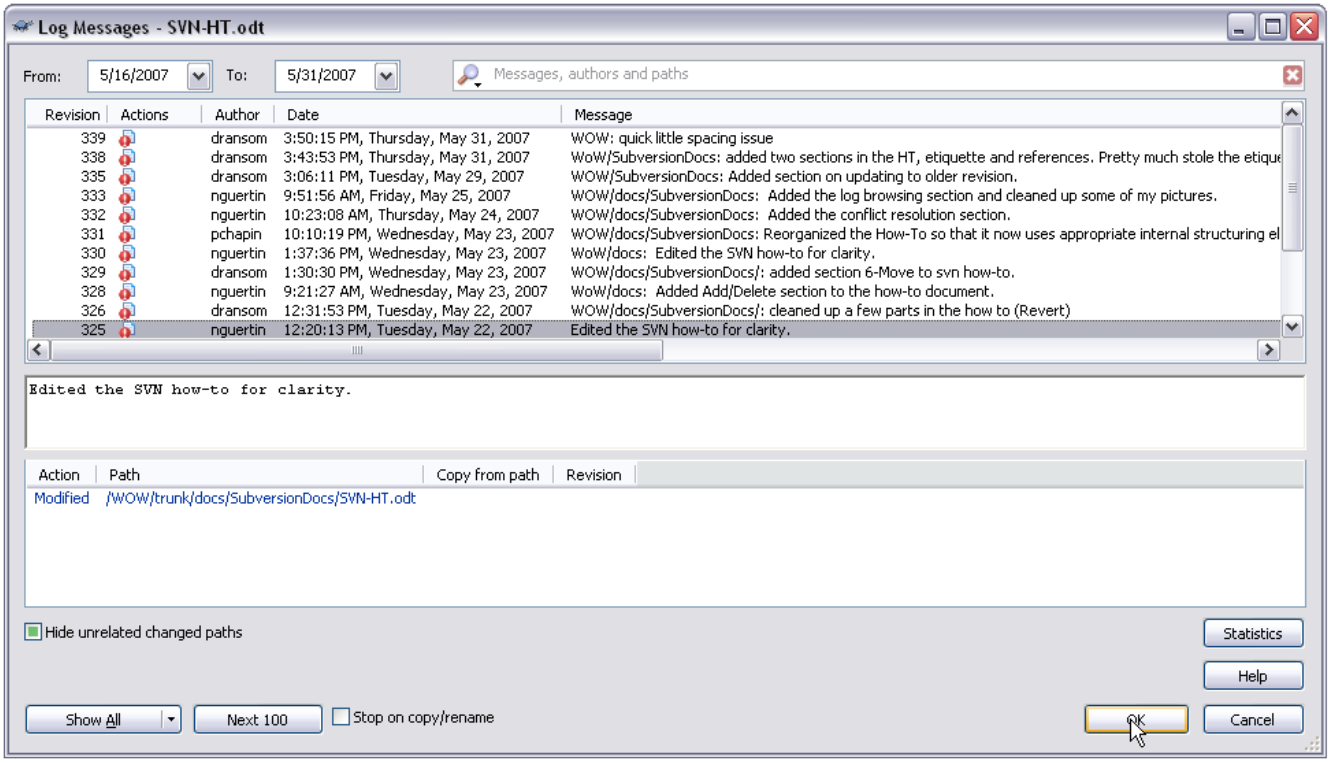
1. Updating your working copy to an older revision can be useful if you want to back-track your steps and start fresh. This is very similar to the Revert command, but instead of changing to the most current revision on your system, you can get any version that exists in the repository. You may want to do this will an entire folder, or just one file. First, right-click on the version controlled file or folder and click TortoiseSVN->Update to revision...



2. A dialog will pop up like the one shown below and will allow you to update either the HEAD (latest) version or one that you specify. If you do not know the exact version number you wish to revert to, click on the "Show Log" button, and you can choose which revision you want via the log messages in the repository.



3. Select OK, and your file or folder will be changed to a previous version.



Etiquette

Files under Subversion version control are being shared with other developers and thus, unlike with most files stored on your system, making changes to Subversion controlled files is a social act. There are standard expectations in the industry regarding how you should behave with respect to version controlled files. If you don't follow those standards you will certainly annoy other developers.

- **Don't include generated files like object files or executable files in the repository.**

  Instead you should generate them in your working copy and leave them uncontrolled. The problem with including generated files in the repository is that they cause a massive waste of repository space and network bandwidth (to keep synchronized). Every time you recompile your program, for example, you will create new versions of the object files and executables.

  Sometimes people include generated files because one or more developers is unable to generate them due to missing or improperly configured tools. There are times when this is reasonable; however the best solution in this case is to get those missing or improperly configured tools fixed so that all developers can generate the files they need from the sources.

- **Don't include personal scratch files in the repository.**

  Files that contain notes to yourself that are not interesting or useful to other developers should not be added to the repository. Instead leave them uncontrolled in your working copy. Similarly configuration files that contain personal settings should not be in the repository. Each developer will likely want to use different personal settings; it's rude to force your settings on everyone

else.

- **Don't include static files (files that will never change) in the repository.**

  For example, third party documentation does not need to be under Subversion control. Similarly third party libraries that you will never modify should not be under Subversion control. While sharing such files between developers is a good idea, there are other more suitable ways to do that (for example, put the files on a project web site).

- **Don't include the same file in multiple formats.**

  This is really another example of the prohibition against generated files. However, it deserves special attention. Some developers will, for example, write documentation using Microsoft Word and then save the documentation in `.doc`, `.rtf`, and `.pdf` format. The rationale for this is that some developers might not want to use Word and so the other formats are being generated as a convenience. However, doing this is a mistake. The development team should agree on what file formats are going to be used and then every developer should get the tools necessary to manipulate those formats. Additional formats can be generated during the build process for distribution to users, but such files should not be subject to version control.

- **Don't include backup files.**

  Developers used to working without version control get in the habit of making backup copies of files before changing them in significant ways to facilitate rolling back if the changes go wrong. This is usually unnecessary when working under a version control system. The Subversion server remembers every version of every file that was committed to it. Indeed, that is one of the main purposes of the system. If you need to roll back changes that have gone wrong or refer to an old version of a file, simply ask the Subversion server to fetch the version you want.

- **Don't commit multiple, unrelated changes at once.**

  Each significant change that you make is likely to involve multiple files. This is fine and expected. However, you should avoid, if possible, committing logically unrelated changes—even if in the same file—in a single commit operation. If one of the changes turns out to be a mistake, it will be impossible to roll it back without also rolling back the other change. Also making multiple unrelated changes in a single commit makes searching and interpreting log messages harder; it can make it more difficult to find when a particular change was done.

- **Don't commit half made changes—except when you should.**

  Ideally each commit operation should be a logically complete change that can be described with a simple log message. Avoid committing changes that are half baked. *Especially avoid committing programs that don't compile!* Committing changes before they are ready makes interpreting the log messages harder and it makes life harder for the other developers since they will get a non-working, semi-changed system the next time they do an update.

  However... some changes are very large and require a lot of work to stabilize. It may not make sense to wait until such a change is completely finished before committing. For one thing other developers might want to see how the work is going and may want to assist. In cases like this, you should make it clear in the log message that the change is unfinished. Also you should communicate what is happening to the other developers so that everyone is aware that the system will be in a non-working state for a while. In situations like this you might consider creating a *branch* where the large change can be hammered out without impacting the main line of devel-

opment. Read about branches in the Subversion documentation if this idea sounds interesting to you.

- **Do include control files necessary for generating the generated files.**

  For example, makefiles, if you are using them, should definitely be in the repository. Any configuration files that affect the way your project is built should all be in the repository. When a developer checks out the project for the first time, that developer should be able to build the project exactly the same was as everyone else (assuming he or she has the right tools installed), using all the same options, without having to build or change any configuration files.

- **Do include scratch files that contain information of interest to multiple developers.**

  While some scratch files might contain information only of interest to you, other scratch files might contain information of interest to many. A "to-do" list is one example of such a file. Files that multiple developers will want to read *and* update should definitely be under version control.

- **Do include any documentation that needs to be shared between developers and that will evolve and change as the project proceeds.**

  Your project's documentation is a development effort just like the code and should be treated accordingly. Thus documentation files should be under version control (unless they are generated from the program source files). You might also consider adding your project's web pages to the repository. You can then just check out that portion of the project to a web server to publish it.

- **Do communicate with other developers.**

  No version control system can replace team communication. Subversion is able to merge changes made by different developers to the same file, however it will report a "conflict" and require manual merging if the changes are both to the same part of the file. In addition, Subversion can do nothing to protect you against high level inconsistencies. For example if two developers try taking a project in two totally different directions, Subversion will obviously have no knowledge about that. As in any team project, coordination between developers is a must if chaos is to be avoided. Using Subversion, or any other version control system, does not change that.

- **Do write high quality log messages.**

  Although it may seem during normal development that the log messages are pointless, you will rely on them greatly if you need to dig back in the repository looking for an old version of some file. For example you might want to find the most recent version of a file *before* a certain change was made. Without good, clear log messages such a task is difficult.

  My suggestion is to prefix your log message with the name of the project. For example a commit to the RTWho project might have a log message such as "RTWho: Fixed the annoying refresh bug with the configuration dialog box (bug #1234 in Bugzilla)."

- **If you need to remove a controlled file, be sure to do it using Subversion so that it gets properly removed from the repository.**

  If you simply delete a controlled file it will return the next time you do a `svn update` operation.

- **If you want to rename a controlled file, be sure to do it using Subversion so that it gets**

properly renamed in the repository.

You can rename a file by removing it under the old name and then adding it back under the new name. However, it is better to use the `svn rename` operation. That way Subversion knows the new name is related to the old file and the logging and version tracking will be more useful.

Command Line Client

Although this document has focused on using the Windows client, TortoiseSVN, Linux users will most likely be using the command line client instead. In fact, the command line client is also available for Windows, both as a native application and as part of Cygwin. Some tools (IDEs, etc) provide Subversion support by invoking the command line client in the background and thus having such a client installed on a Windows system is not unusual or useless.

The command line client is called `svn`. It supports a large number of "subcommands." Use the command:

```
$ svn help
```

For a list of the subcommands and their arguments. Note that the `$` character above (and in the examples below) represents the command prompt. To check out a working copy use a command such as:

```
$ svn checkout svn://svn.ecet.vtc.edu/Scratch/trunk Scratch
```

The last argument is the name of the directory where the working copy will be placed. If that argument is omitted `svn` will use the last component of the repository URL (`trunk` in the example above) instead. Often that is fine, but in this case it seems undesirable to put the working copy in a directory named `trunk`. If the working copy directory does not yet exist it will be created. In fact, this is the common case.

All other operations on a working copy should be done with the root of the working copy as the current directory. The `svn` client applies each operation to all subdirectories recursively. To update a working copy use the command:

```
$ svn update
```

To find out which controlled files, if any, have been modified locally use the command:

```
$ svn status
```

To commit changes to the repository. Use the command:

```
$ svn commit -m "This is a log message."
```

Despite its modest appearance the command line client is every bit as powerful as the TortoiseSVN client. The commands above are the most commonly used and provide the basic services one normally needs. However don't hesitate to consult the Subversion documentation for more details on `svn`'s many options.

References:

If you wish to learn more, here are a few resources that we can recommend:

1. Practical Subversion 2nd ed – Daniel Berlin and Garrett Rooney. Apress. http://www.apress.com/

book/bookDisplay.html?bID=10203. ISBN – 1590597532

2. Pragmatic Version Control 2nd ed – Mike Mason. The Pragmatic Programmers. http://www.pragmaticprogrammer.com/titles/svn2/index.html ISBN – 0-9776166-5-7
3. Subversion Homepage – http://subversion.tigris.org/