



## **Configuring a Development Environment with Apache, Subversion, TortoiseSVN, and Subclipse**

Aaron West  
Adobe Community Expert  
Nashville ColdFusion User Group Manager  
Certified Advanced Adobe ColdFusion MX 7 Developer  
<http://www.trajiklyhip.com/blog>  
March 11, 2007

# Table of Contents

<b>Introduction.....</b>	<b>3</b>
About Me.....	3
Why Subversion Over CVS and Microsoft Visual SourceSafe?.....	3
Disclaimer.....	4
<b>Part 1: Installing and Configuring Apache 2.0.59.....</b>	<b>5</b>
Advantages of Using the Apache Web Server.....	5
Downloading Apache.....	5
Installing Apache.....	5
Apache Basics.....	8
Changing Apache's Listen Port.....	10
Summary.....	11
<b>Part 2: Installing Subversion 1.4.2 and Integrating Subversion With Apache.....</b>	<b>12</b>
Downloading Subversion.....	12
Installing Subversion.....	12
Verify the Subversion Installation.....	15
Configure Apache and Subversion Modules.....	15
Creating the Repositories Directory.....	17
Exposing the Repositories Directory to Apache.....	17
Testing the Initial Subversion – Apache Hook.....	18
Enabling Apache Virtual Hosts.....	19
Configuring a Virtual Host for Subversion.....	19
Creating the Custom Logs Directory.....	20
Testing the Final Subversion – Apache Hook.....	21
Summary.....	21
<b>Part 3: Installing TortoiseSVN 1.4.3 and Creating Your First Repository.....</b>	<b>22</b>
What is TortoiseSVN?.....	22
Downloading TortoiseSVN.....	22
Installing TortoiseSVN.....	22
Creating Your First Repository.....	26
Download the Sample SQL Code.....	31
Import Sample SQL Code to the Trunk.....	32
Creating a Working Copy of the Repository.....	34
Committing Changes to the Repository.....	36
Getting Changes from the Repository.....	37
Summary.....	38
<b>Part 4: Subversion and Apache – Better Logging and Authenticated Access.....</b>	<b>39</b>
Default Apache Logging.....	39
Custom Apache Logging.....	39
Basic Apache Authentication.....	40
Summary.....	43
<b>Part 5: Accessing Subversion Repositories With Subclipse.....</b>	<b>44</b>
What is Subclipse?.....	44
Installing Subclipse.....	44
Creating an Eclipse Project Connected to a Subversion Repository.....	50
Committing Changes to the Repository.....	55
Getting Changes from the Repository.....	57
Synchronizing a Working Copy with the Repository.....	58
Summary.....	58
<b>Additional Resources.....</b>	<b>59</b>
Books.....	59
Links.....	59
Software.....	59

## Introduction

Over the past two years I've seen the popularity of Source Control Management (SCM) increase exponentially. I've yet to determine exactly what the catalyst has been but certainly overall awareness regarding the importance of SCM has played a major role. During sessions at various ColdFusion conferences and user group meetings, audiences have been polled regarding their use of source control and over the years more hands are being raised as development teams begin to employ some level of SCM. It's uplifting to see this change and yet there is still room for improvement.

Sprinkled throughout the Internet are several blog postings and articles designed to help developers create and configure a source control environment. Most of these resources are out of date or incomplete when it comes to illustrating an end-to-end solution. That's not to say they aren't helpful but in my own endeavor to set up a new source control environment I've found them somewhat wanting. So, I decided to create a new resource for installing and configuring a source control environment. Throughout the process I've researched tons of different options, documenting what I think works best for my team. Several dozen hours have gone into installing, configuring, re-installing and re-configuring in an effort to get things "just right" and along the way I've documented the entire process. While this was done in forethought of this text it has also been helpful to create a paper trail of what I've done. What follows this introduction are detailed instructions for installing and configuring the same source control environment I've created, complete with dozens of screen shots. Before moving forward I should note this text focuses on both the server and client aspect of source control. Not only will I cover the installation and configuration of Apache and Subversion as server software; I also cover the installation and use of various Subversion client tools like TortoiseSVN and Subclipse. While the information in this text is meant to be consumed as a complete whole you can certainly focus on the portions relevant to you.

## About Me

<sup>1</sup>Aaron West has been working in the Web development space for over seven years. He holds a bachelor's degree in Computer Science & Information Systems and has worked for several firms in the Nashville area in the e-learning, training, and automotive arenas. Aaron currently serves as the Development Manager at Dealerskins - a leading provider of Web applications and Web services to the automotive industry - heading a team of developers as they build the most beautiful, powerful, and distinctive Web solutions possible.

Extremely passionate about the Web, Aaron has written several articles and tutorials on ColdFusion and Flash, and integrating these technologies with various database platforms. He also participated in the Deitel, Deitel & Associates book *Internet & World Wide Web Programming 3rd Ed.* as a technical reviewer. As an Adobe Community Expert he shares his technical expertise with the world-wide community through high-caliber peer-to-peer communication. He's also a Certified Advanced Adobe ColdFusion MX 7 Developer, a certified Adobe Flash Developer, and he manages the Nashville ColdFusion User Group.

Other than immersing himself in all things technology, Aaron enjoys spending time with his family, riding his sportbike, reading, and exploring the underwater worlds around us. He is available for speaking engagements and conferences and can be reached at [aaron@trajiklyhip.com](mailto:aaron@trajiklyhip.com).

## Why Subversion Over CVS and Microsoft Visual SourceSafe?

First and foremost, yes there are options other than [Subversion](#), [CVS \(Concurrent Versions System\)](#), and [Visual SourceSafe \(VSS\)](#). With the exception of perhaps [Perforce](#) I've found these three to be the most popular source control products. I spent the better part of three years using VSS at my first job. Back then, there was no CVS or Subversion and Microsoft was arguably the largest player in SCM. While VSS did help the development team manage code, it did so with an insurmountable laundry list of headaches. The VSS client was not intuitive and required developers to mimic folder structures that were already present on the file system. This "repository view" is something developers should not have to worry about or understand.

---

<sup>1</sup> Bio copied from <http://www.adobe.com/communities/experts/members/143.html>

Managing binary files like images and Flash movies with VSS was horrific to say the least and we gave up after a half dozen failed attempts. The last, and probably largest pitfall of Visual SourceSafe was its inability to allow multiple developers to work on the same code at the same time. The process of changing versioned files included “checking out” the code. Once checked out the code was locked and modifications by other developers were prohibited. If the first developer was working on a version 2 release, bug fixes (by other developers) for version 1 were not allowed.

At another company I had the opportunity to use CVS. When compared with VSS, CVS was a fresh breath of air. It was much easier to use (we used the WinCVS client) than VSS and it handled binary file versioning like a champ. In fact, Flash files (SWF's and FLA's) were what we versioned most. My only real complaint with CVS was its use of a *lock-modify-unlock* model like VSS. This model requires a user lock a file for modification – locking out other users and changes – and then unlock the file after changes are complete.

Subversion employs a *copy-modify-merge* model as an alternative to locking. This model allows developers to work on a local copy of files (called working copies) simultaneously and independently. Developers may then commit their work to the versioned repository where Subversion will merge the changes from each developer into a final copy. Other benefits of Subversion include the ability to integrate with the [Apache Web server](#), versioning of directories and directory/file renaming, and cheap branching and tagging operations. Since a full discussion of SCM and SCM policies is outside the scope of this text, I recommend reading [Chapter 1. Fundamental Concepts](#) of the [Subversion Book](#).

### *Disclaimer*

There are many, many ways SCM can be employed. How you use SCM should be determined by your own needs and the needs of your team. This text describes one such way of utilizing source control and in no way represents an applicable solution for all environments. I encourage you to use my ideas as a guide in discovering what works best for you and your team.



## Part 1: Installing and Configuring Apache 2.0.59

Part 1 of this text is going to walk you through installing and configuring the Apache Web server in preparation for hooking Apache and Subversion together.

### *Advantages of Using the Apache Web Server*

Apache is not required in order to run Subversion. Subversion includes it's own standalone server, called *svnserve*, that allows you to deploy and utilize Subversion repositories. The standalone server works just fine and can be installed on just about any operating system out there (Windows, \*nix, and Mac OS X). *Svnserve* offers deployment as an inetd service, a daemon service, offers basic authentication and authorization, can be utilized over an internal network, and can be tunneled over ssh for security. So why integrate Subversion and Apache?

There are several advantages to using Apache especially if your team is already running the Apache Web server. First and foremost, your Subversion repositories have more reach, and are available wherever your Apache Web server is available. Depending on your network configuration this may mean expanded availability for remote developers or developers that need to work from home for a day (ever waited 4 hours for the cable guy?) Another great benefit is enhanced repository access logging through Apache's built-in logging mechanisms, and since we're using HTTP, that means we can also make use of HTTPS for encrypted repository access over SSL. As if that weren't enough you gain built-in repository browsing through your Web browser of choice. Given all the benefits of Apache, you may be wondering why anyone would use the built-in *svnserve* Subversion server. The answer, is simplicity: it's really easy to set up and configure *svnserve* while a bit more complex to set up and configure repository access through Apache. Hopefully, this text will make the latter much less painful for you.

### *Downloading Apache*

First things first, you need to download Apache. For the purposes of this text all references to and illustrations of Apache are of version 2.0.59. However, as long as you are working with 2.x all instructions herein should be accurate. To get Apache, point your Web browser to <http://httpd.apache.org/download.cgi>. This page offers 2.2.x as well as (at the time of this writing) 2.0.59. Since this text focuses on setting up a Windows server with Apache, you'll want to download the Win32 Binary which should look something like: `apache_2.0.59-win32-x86-no_ssl.msi`.

### *Installing Apache*

To begin installing Apache you'll need to find and double-click the \*.msi file you downloaded earlier. You might also want to think about *where* you are installing Apache. All the screen shots referenced in this section refer to a development server installation. This is the most common practice on small to large teams but if you're an individual developer you may be installing everything on your personal machine. Either way, the process is the same; just be aware of the machine differences as you walk through the server installations (Apache, Subversion, and TortoiseSVN) and the client installations (TortoiseSVN and Subclipse).

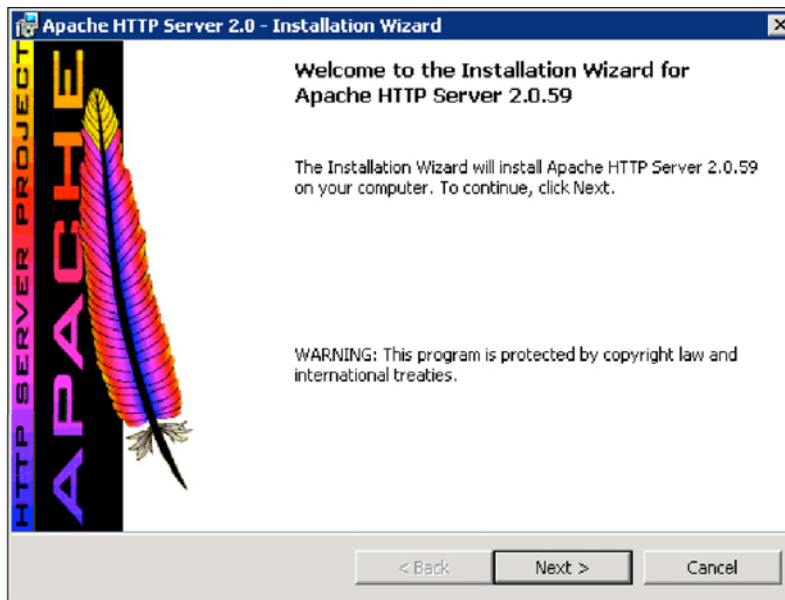


Figure 1: Begin Apache Installation

Once the initial Apache installation screen displays, press *Next* to continue.

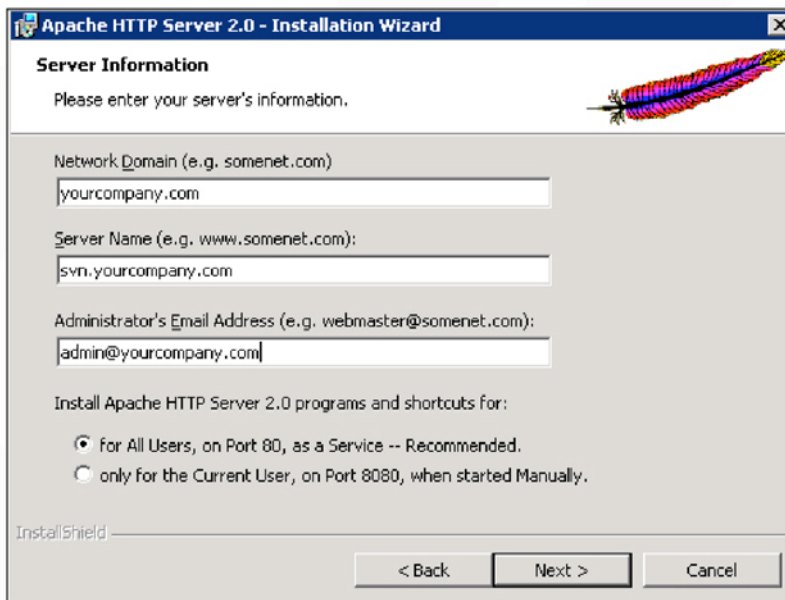


Figure 2: Configure Basic Apache Settings

In this screen you'll configure some basic settings for your Apache server. If you are installing Apache on a development server you will likely use some sort of company or personal domain. If you are installing Apache on an individual development computer you will probably use 127.0.0.1 for both the *Network Domain* and the *Server Name*. Just remember what you put here, because all work from now on will use these initial settings. If you follow my example to the strictest detail (as will the rest of this text) you will need to ensure the *Network Domain* you enter points to your development server. This is accomplished – most often – with DNS entries on your corporate network. If these entries are not in place you can edit your personal computer's *hosts* file to point the *yourcompany.com* and/or *svn.yourcompany.com* domains to your development server. One last note, leave the radio button set to *All Users, on Port 80* for now. I'll demonstrate how to change this later in order to provide a very singularly-focused Apache/Subversion installation.

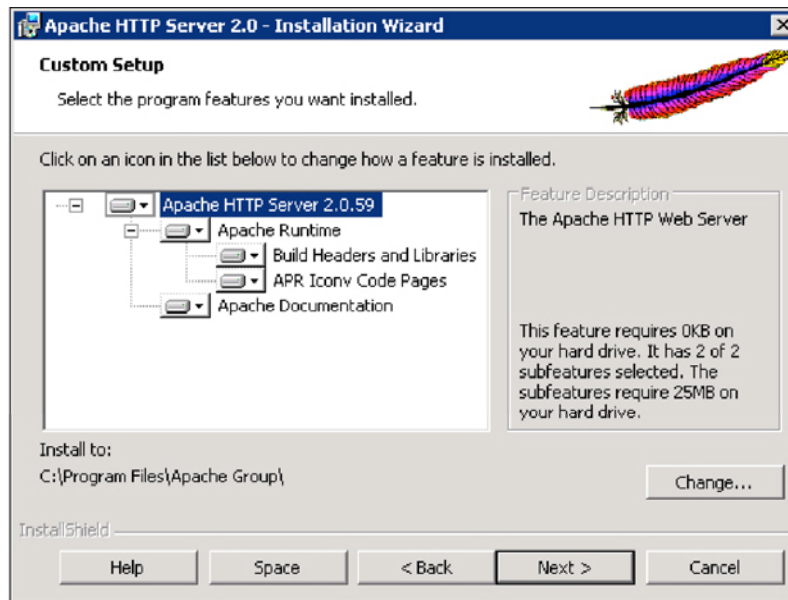


Figure 3: Custom Setup Screen

The Custom Setup screen is designed to show you specifics regarding what Apache will be installing and where. All of the default installation settings are fine for the purposes of this text.

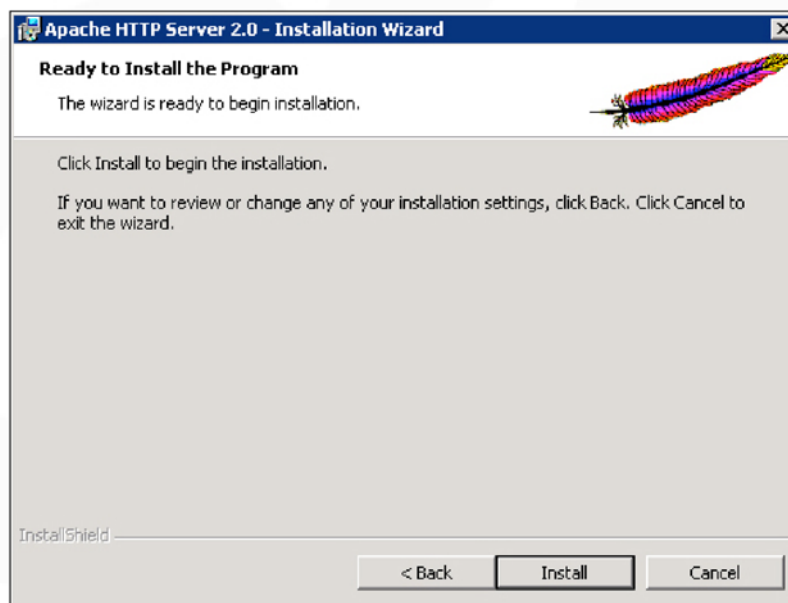


Figure 4: Ready to Install Apache

You're now ready to install Apache. Press the Install button and relax for a few seconds.

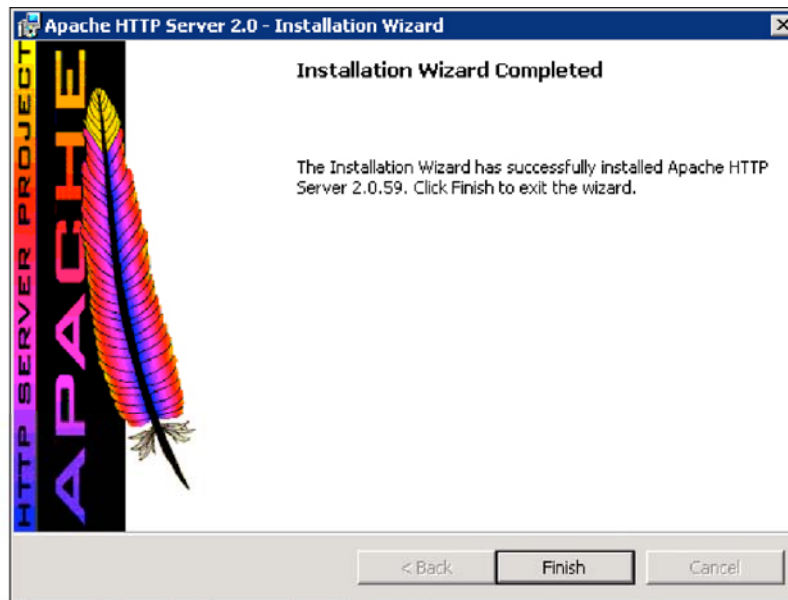


Figure 5: Apache Successfully Installed

Once the installation has finished your screen should look similar to that of *Figure 5*. At this point, Apache is installed and configured to run on port 80 (the default port). If you already had a Web server (like Microsoft's Internet Information Server) running on port 80 Apache may not be running. As part of this text I will be configuring Apache to run on port 81 as a dedicated Subversion Web server. This is a requirement in my development environment as IIS is already running on port 80. Thus, all examples and screen shots from here on will have port 81 in them, but, let's not get ahead ourselves. Before making this change let's get familiar with three important pieces of the Apache Web server: the htdocs (webroot) directory, the conf (configuration) directory, and the Apache Monitor.

## Apache Basics

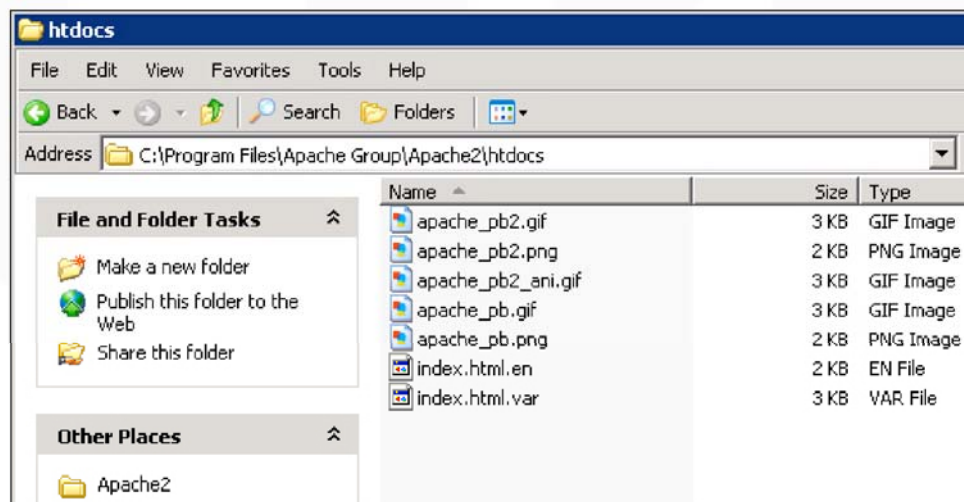


Figure 6: The Apache Web Root Directory

By default, Apache is installed to `C:\Program Files\Apache Group\Apache2`. Apache's Web root is, by default, located in the `htdocs` directory. The files shown in *Figure 6* are all that is needed for Apache's test page (which we will be viewing shortly) to function properly. All other files are for internationalization and can be safely removed.

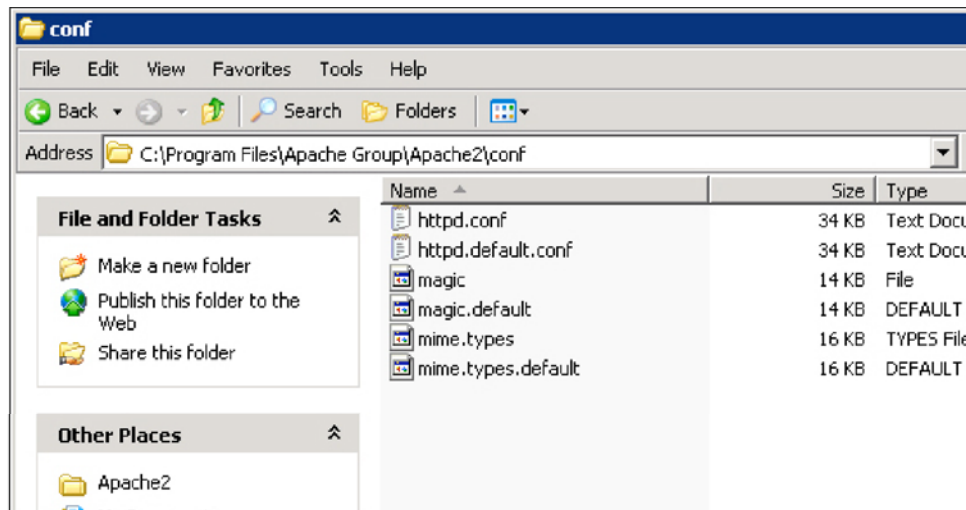


Figure 7: The Apache Configuration Directory

Apache's configuration is determined by the settings in the *httpd.conf* file. For convenience and safe keeping Apache has already created a backup of this file for you (*httpd.default.conf*). Should you ever mess something up in the *httpd.conf* file you can always revert back to the original, factory settings, by making a copy of *httpd.default.conf* and naming the copy *httpd.conf*. It's also a good idea to make a habit of backing up your *httpd.conf* file just before making major changes. I'll be directing you to make numerous changes to the configuration file as we progress through the various parts of this text. For now, let's take a look at the Apache Monitor.

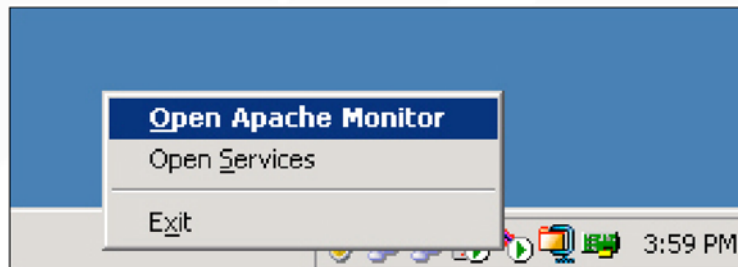


Figure 8: Accessing the Apache Monitor from the System Tray

The quickest and easiest way to view the *Apache Monitor* is to right-click the *Apache* icon in the system tray and select *Open Apache Monitor*.



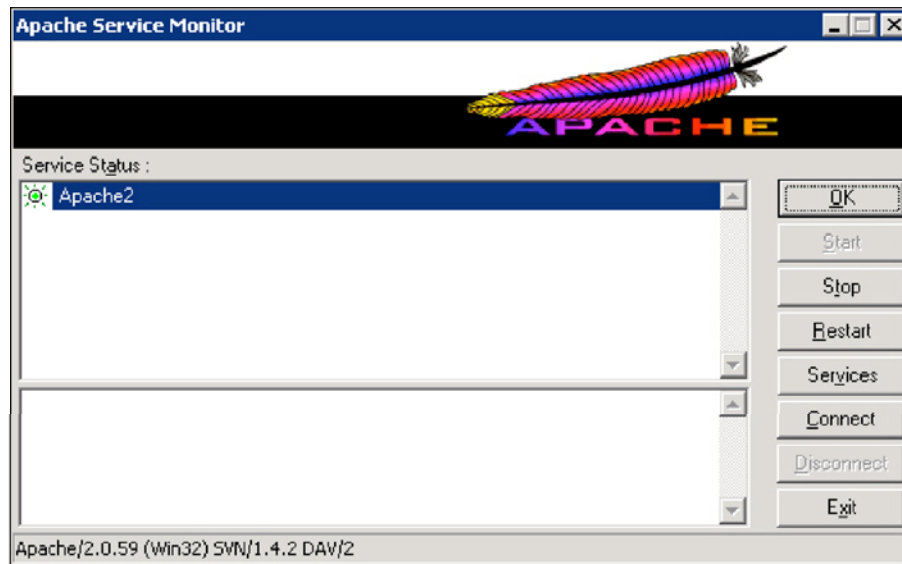


Figure 9: The Apache Service Monitor

The *Apache Service Monitor* (or *Apache Monitor* for short) is your central location for starting and stopping the Apache Web service. I'll reference the *Apache Monitor* extensively as we move through this text. Now that we've gotten familiar with the *htdocs* directory, the *conf* directory, and the *Apache Monitor*, let's make one or two changes before we test Apache to make sure it's working properly.

### Changing Apache's Listen Port

```
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#
#Listen 12.34.56.78:80
Listen 81
```

Figure 10: Editing the Apache Listen Port

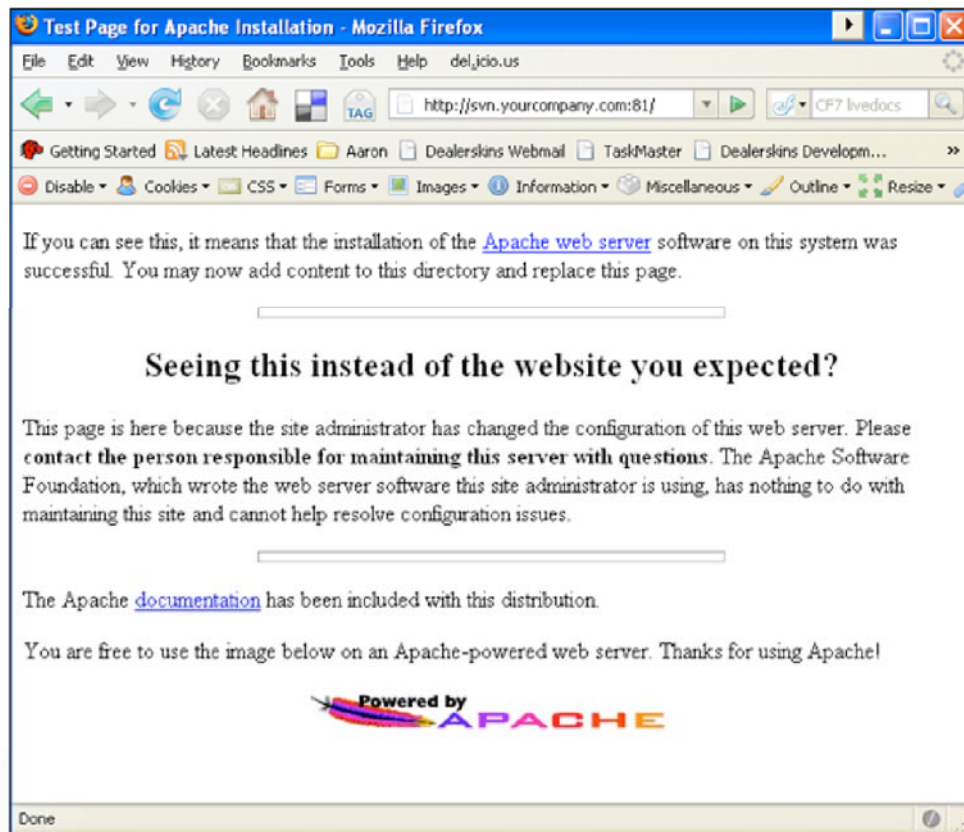
Earlier I mentioned my development team server runs Internet Information Server on port 80. Because of this, I must run Apache on a different port. Port 81 is a nice alternative and will allow my Apache Web server to run alongside of IIS. You don't have to perform this step but all screen shots and references to Apache will include use of port 81. Perform all subsequent steps based on your configuration. To change the listen port you'll need to stop Apache using the *Apache Monitor*. Then, navigate to Apache's configuration folder (Figure 7) and open the *httpd.conf* configuration file (Notepad works great), find the *Listen 80* text and change it to read *Listen 81*.

```
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address anyway, and this will make
# redirections work in a sensible way.
#
ServerName svn.yourcompany.com:81
```

Figure 11: Setting Apache's ServerName

Since I've changed the port Apache is listening to I want to make sure Apache's *ServerName* reflects the same information. Doing this is completely optional but I like having all places that display the *ServerName* remind me of the configuration. To change the *ServerName*, find the appropriate line in the *httpd.conf* configuration file and change it so it matches Figure 11 above. Make sure you save your changes and start Apache using the

*Apache Monitor*. If there are any problems with what you've changed, the *Apache Monitor* will be happy to notify you when you try and start up the server. If you do experience problems, double-check your work and the screen shots above and try restarting Apache again.



With Apache up and running and after altering the *Listen Port* and *ServerName*, it's time to test our changes. Open your Web browser and point it to *svn.yourcompany.com:81* replacing "yourcompany" with the domain you entered during the installation of Apache (Figure 2). If all goes well you should see a display exactly like *Figure 12* above. If the page comes up but doesn't look quite right, it's possible you removed too many files from the *htdocs* directory (Figure 6). Verify your *htdocs* directory has at least those files in *Figure 6* and reload the page. If you get an error indicating there's no Web server running or the Web server can't be found you need to make sure the domain you entered during installation (Figure 2) is pointing to your server's IP address.

## Summary

Let's review our work. We started by installing Apache using all the default settings. I then discussed the Apache *htdocs* and *conf* directories and the *Apache Services Monitor*. Next, we made a few changes to our Apache configuration (*httpd.conf*) directing Apache to listen to port 81 instead of port 80. We also changed the Apache *ServerName* so that indicates Apache is listening to port 81 instead of the default. Last but certainly not least, we tested our new Apache Web server by loading its default page in a Web browser. With all of this out of the way, it's time to install Subversion and integrate it with Apache!

## Part 2: Installing Subversion 1.4.2 and Integrating Subversion With Apache

In Part 1 I walked through the installation of the Apache Web server. With that step complete it's time to install the Subversion server and integrate Subversion and Apache. Once we've completed these steps we'll have our source control server environment in place and we'll be ready to install some Subversion client tools.

### *Downloading Subversion*

Several resources on the Internet offer instructions on downloading and installing a Subversion server. Many of the resources focus on ease of installation and recommend the [Subversion 1-click installation](#). This installer does make things easy by installing the Subversion command-line utilities (*svnserve*), and TortoiseSVN. It also creates an initial repository and project. We will not be using the 1-click installer and there are several reasons why. First, the 1-click installer sets up *svnserve*, the standalone Subversion server, and I'm focusing specifically on integrating Subversion with Apache. With Apache integration your Subversion server and its repositories are available as long as Apache is running. In other words, the standalone server (*svnserve*) is not used. If you were to not integrate Subversion with Apache you would have to start and stop the *svnserve* service explicitly. For more advantages regarding Subversion and Apache over *svnserve*, check out the *Advantages of Using the Apache Web Server* section in Part 1.

To download the Subversion server point your Web browser [here](#). This Web page lists several Windows versions of Subversion most notably, the Win32 binaries. At the time of this writing both 1.3.2 and 1.4.2 were available for download. For the purposes of this text, you should download *svn-1.4.2-setup.exe*. **UPDATE: Just before releasing this paper, version 1.4.3 of Subversion was publicly released. While this text remains centered around 1.4.2, you should have no problems installing and using 1.4.3.**

### *Installing Subversion*

Just like the Apache installation, Subversion should be installed on your development server. To begin installing Subversion double-click the executable you downloaded earlier.

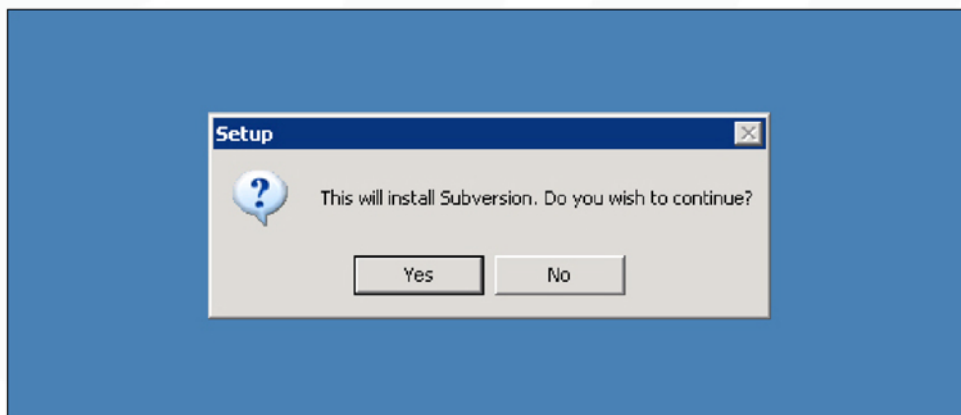


Figure 13: Begin Subversion Installation

Subversion prompts you to confirm you want to install the software.



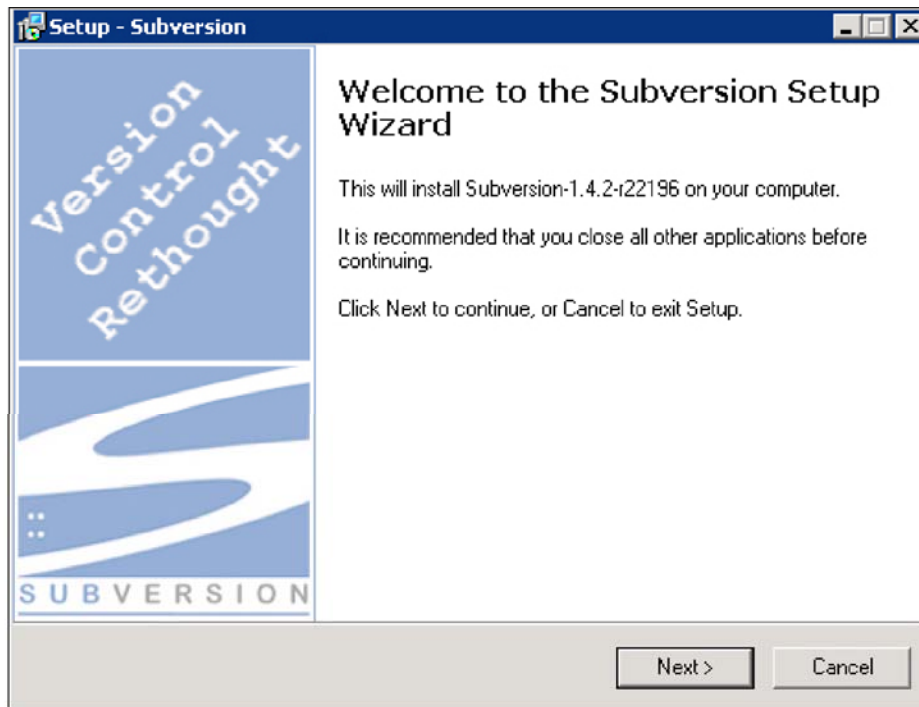


Figure 14: Subversion Installation Welcome Screen

The Subversion installation lets you know which version (and build) you are installing. Press *Next* to view the software terms and license. Accept the terms and press *Next*.

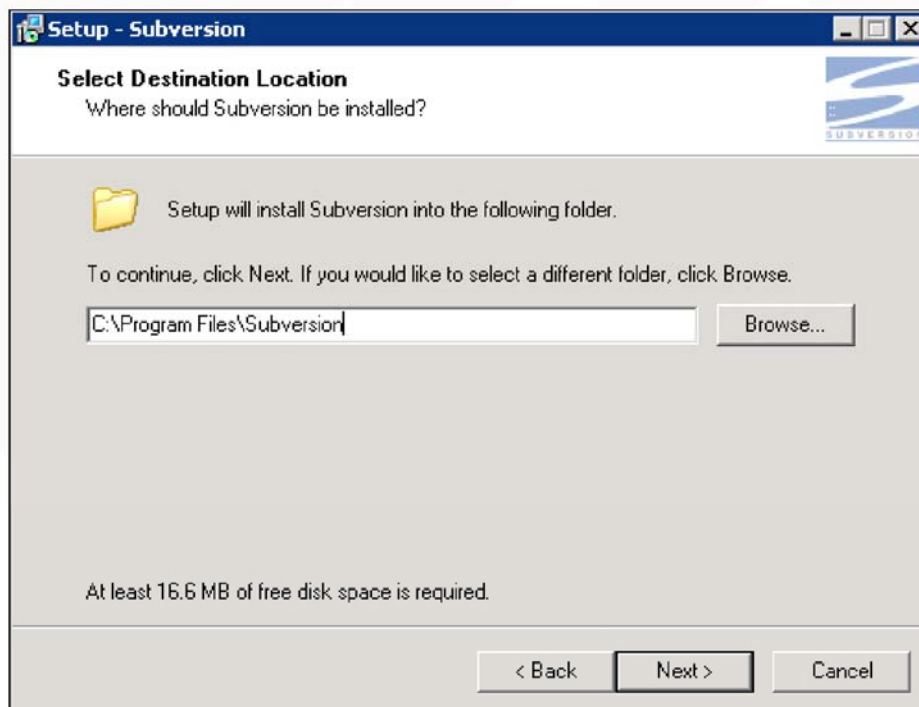


Figure 15: Subversion Destination Location

The default destination directory is fine and is what I reference throughout this text.

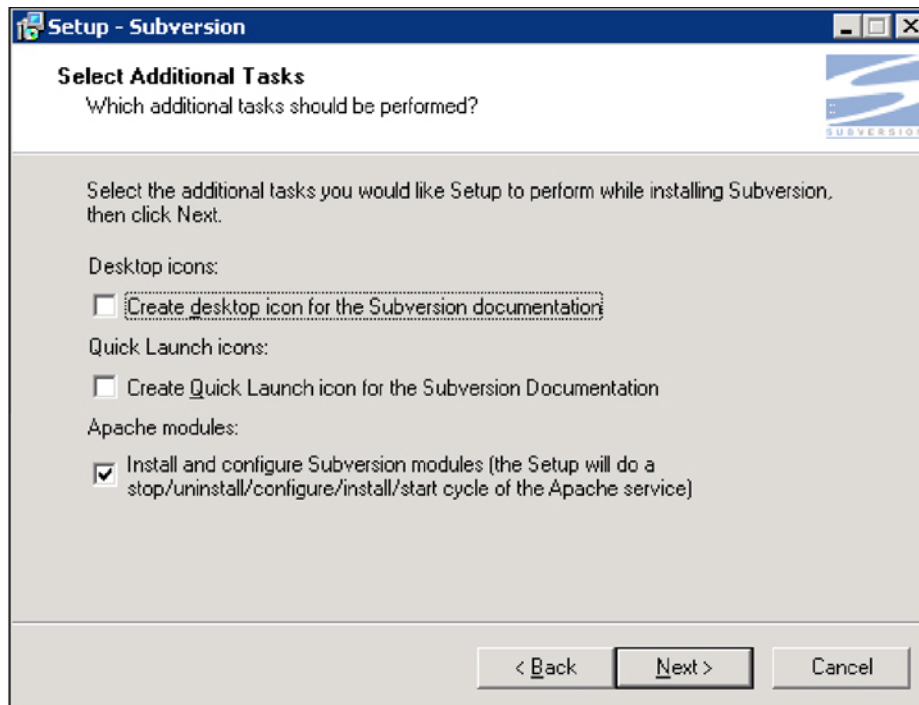


Figure 16: Additional Subversion Installation Tasks

Select which additional installation tasks you want the installer to perform. Make sure you select the last option, which directs the installer to add a few “hooks” to your Apache Web server by editing your *httpd.conf* configuration file.

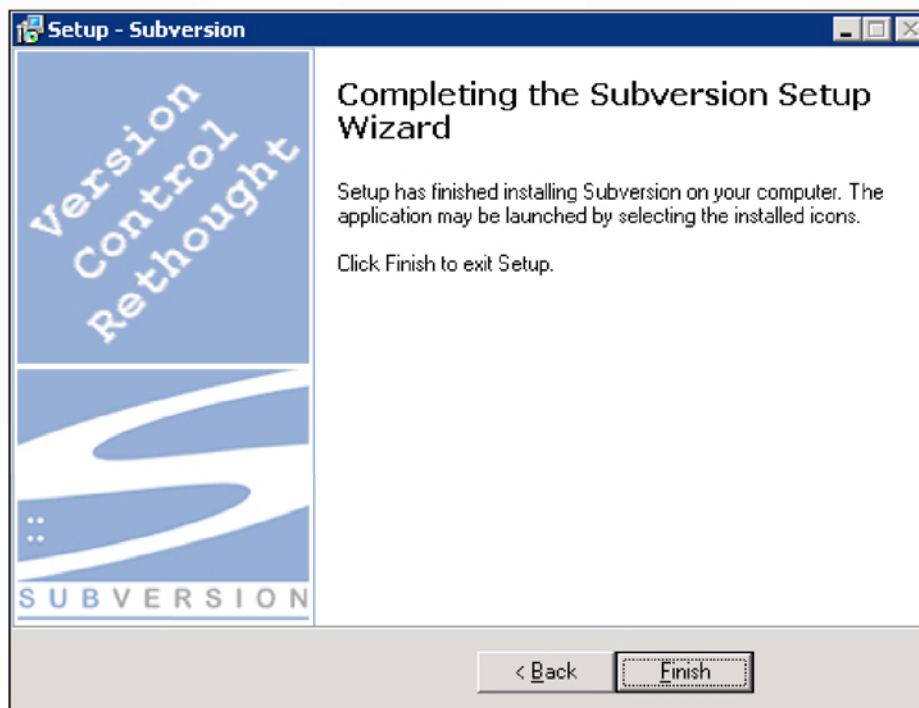


Figure 17: Subversion Installation Completion

After Subversion has been installed you'll see a completion screen. Press *Finish* to exit the installation.

## Verify the Subversion Installation



Figure 18: Verifying the Subversion Installation

Before we move further we need to verify Subversion was installed correctly by making sure the “hooks” were added to Apache. One way to do this is to request an invalid Web page (`index.html` is not in the Apache Web root - see Figure 6) from Apache since requesting a page that doesn't exist will cause Apache to display the default “Not Found” page. This page has, at the bottom, information pertaining to the version of Apache running on our development server as well as the version of Subversion we just installed. In Figure 18, the key text is *SVN/1.4.2 DAV/2 Server*.

## Configure Apache and Subversion Modules

```
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so

#ASw: Added a blank line above and below to make the following line
LoadModule dav_module modules/mod_dav.so

#LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
#LoadModule file_cache_module modules/mod_file_cache.so
#LoadModule headers_module modules/mod_headers.so
LoadModule imap_module modules/mod_imap.so
LoadModule include_module modules/mod_include.so
```

Figure 19: First Change to `httpd.conf`

There are three changes the Subversion installation made to Apache's `httpd.conf` configuration file – all of which deal with Apache's module API. Apache modules are external Shared Objects (.so file extension) that can be included or “loaded” into Apache when the Web server starts. The `LoadModule` directive is pretty simple, mapping a module name to its equivalent .so file on the server. By default, certain modules are loaded while others are not. Modules that are not loaded are commented out in the `httpd.conf` file with a pound (#) sign. To include a module, and its functionality, stop the Apache server, find the `LoadModule` directive for the module you want to include, and remove the pound sign. Figure 19 illustrates Subversion's first change, uncommenting the DAV module (`mod_dav.so` located in Apache's `modules` directory). I've added a blank line

before and after the DAV module to make the line stand out in the screen shot. While a full discussion of the *mod\_dav.so* library is outside the scope of this text a basic explanation is in order. The *mod\_dav.so* module provides <sup>2</sup>WebDAV (Web-based Distributed Authoring and Versioning) functionality for Apache. WebDAV extends the stateless HTTP protocol giving Apache the ability to create, move, copy, and delete files and directories on the Web server. You can think of *mod\_dav.so* as the central hub to all Subversion actions operating on the repositories you create.

```
#LoadModule spelling_module modules/mod_spelling.so
#LoadModule status_module modules/mod_status.so
#LoadModule unique_id_module modules/mod_unique_id.so
LoadModule userdir_module modules/mod_userdir.so
#LoadModule usertrack_module modules/mod_usertrack.so
#LoadModule vhost_alias_module modules/mod_vhost_alias.so
#LoadModule ssl_module modules/mod_ssl.so

#ASw: Added a blank line above and below to make the following line stand out.
LoadModule dav_svn_module "C:/Program Files/Subversion/bin/mod_dav_svn.so"
LoadModule authz_svn_module "C:/Program Files/Subversion/bin/mod_authz_svn.so"

#
# ExtendedStatus controls whether Apache will generate "full" status
# information (ExtendedStatus On) or just basic information (ExtendedStatus
# off) when the "server-status" handler is called. The default is off.
#
#ExtendedStatus On
```

Figure 20: Second Change to *httpd.conf*

The second and third changes the Subversion installation made are the addition of two new *LoadModule* directives. These are placed at the end of all other *LoadModule* directives and are visible in Figure 20. The first module, *mod\_dav\_svn.so* is used to make repositories available to others over a network and works hand-in-hand with *mod\_dav.so*. The second module, *mod\_authz\_svn.so* is used to enable per-directory access control over your repositories. In order to get our *httpd.conf* file cleaned up there are a few changes we need to make. First, you'll notice the *mod\_dav\_svn* and *mod\_authz\_svn* modules are located in Subversion's bin directory. There's nothing inherently wrong with this but since all the rest of the Apache modules are located in Apache's modules directory, I recommend moving these files to *C:\Program Files\Apache Group\Apache2\modules\*. To do this, stop Apache using the Apache Monitor. Move the *mod\_dav\_svn.so* file to Apache's modules directory. You don't need to move the *mod\_authz\_svn.so* file as I won't be converging per-directory access controls. Now, make a few changes to the *httpd.conf* file. First, delete the *LoadModule* directive for *mod\_authz\_svn* and then move the *mod\_dav\_svn* directive from its current location to the line immediately following the *mod\_dav* *LoadModule* directive. It's imperative the *mod\_dav\_svn* directive be located after the *mod\_dav* directive as it depends on it being loaded first. Finally, edit the directory path for *mod\_dav\_svn* changing it to *modules/mod\_dav\_svn.so*. When you're done, your *LoadModule* directives should look just like those in Figure 21.

```
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so

#ASw: Added a blank line above and below to make the following line :
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so

#LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
```

Figure 21: Final *LoadModule* Directives

With the two *LoadModule* directives configured (Figure 21) we have Apache and Subversion talking to each other. However, we aren't ready to start Apache yet as we haven't told it where our Subversion repositories are

<sup>2</sup> For more on WebDAV see [http://www.webdav.org/mod\\_dav/](http://www.webdav.org/mod_dav/)



located nor have we created a directory to hold our repositories!

### *Creating the Repositories Directory*

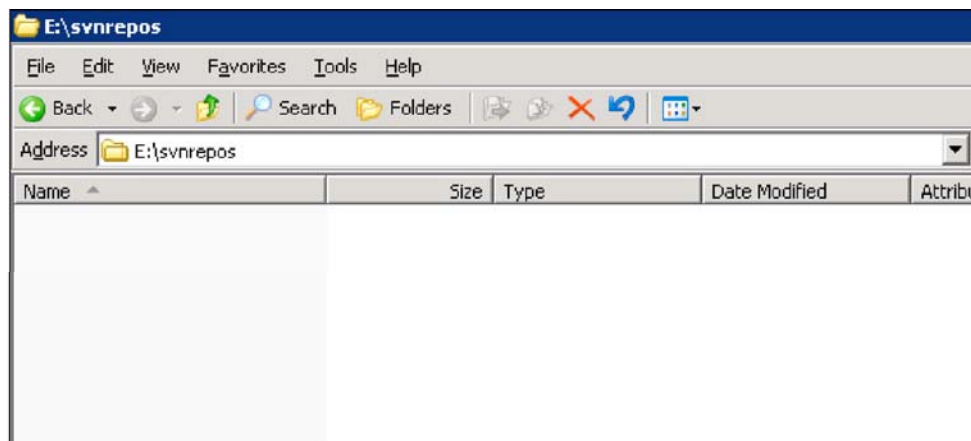


Figure 22: Create the repository directory.

Subversion repositories must be stored somewhere on the server and it's common practice to place them in a *svnrepos* folder. This is usually created at the root of the C: drive but in my implementation it's the root of the E: drive. It doesn't really matter where you create this folder as long as you remember the location and make the appropriate *httpd.conf* changes as we progress. Subversion clients (we haven't gotten to this part yet) will interact with Apache which will in turn interact with the folder and files that (eventually) live in the *svnrepos* directory. One final thing to mention about this folder; it is for Subversion *only*. You shouldn't ever modify, move, delete or directly alter its contents. The only process that should do so is the Subversion server.

### *Exposing the Repositories Directory to Apache*

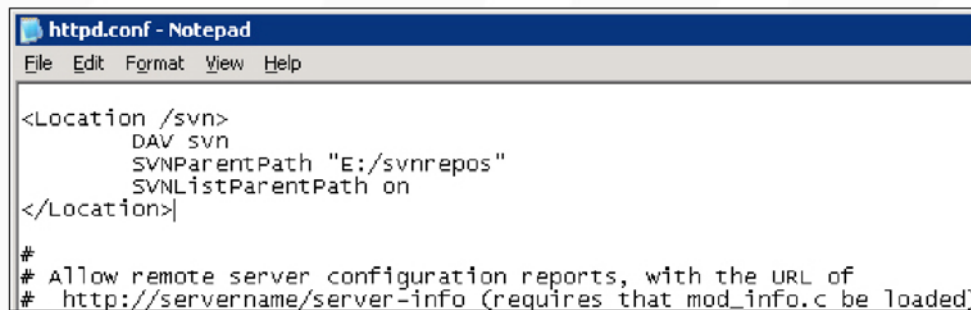


Figure 23: Exposing the repository directory to Apache

In order for Apache to know where our repository directory is located we must give it some instructions. This can be accomplished several different ways but we'll start with the *Location* directive. This directive gives Apache specific instructions when requests are received for a named resource (a URL path). For Subversion access, we simply want Apache to hand off requests for URLs that point at versioned resources to the DAV layer. This is accomplished by mapping a URL path to the absolute directory path of the repository. In Figure 23 above, the URL path is */svn* and the absolute directory path of the repository is the directory we created earlier (E:/svnrepos) with the slash reversed<sup>3</sup>. This path translates to *http://svn.yourcompany.com:81/svn* for our purposes. The *DAV svn* directive tells Apache's *mod\_dav* to use the Subversion server when any requests come in using the aforementioned URL. The *SVNParentPath* represents the absolute location of our repository directory. This directive is special in that it informs Apache any child directories in the nominated path contain

<sup>3</sup> Apache requires absolute directory paths in the form *path/to/directory*.

Subversion repositories. Another option is to use the *SVNPath* directive which tells Apache where the Subversion repository is located without giving it information on child directories. Either *SVNPath* or *SVNParentPath* must be present but not both. The last directive, *SVNListParentPath* on is convenient when multiple repositories will be stored in *E:\svnrepos* and browsed via a Web browser. Anytime a user visits *http://svn.yourcompany.com:81/svn* in a Web browser they will see a list of all the repositories on the server. Because of the obvious security risk imposed by exposing all the names of the repositories (and there locations) this setting is turned off by default.

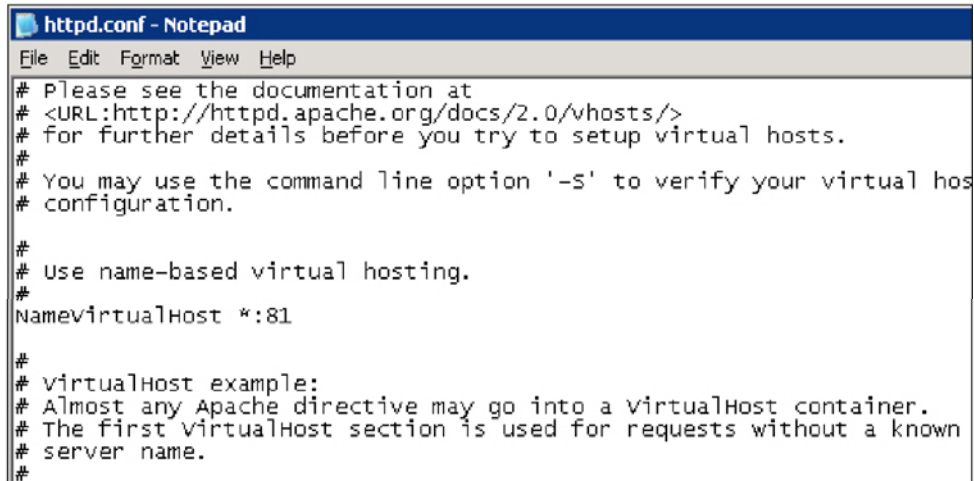
### *Testing the Initial Subversion – Apache Hook*



Figure 24: Testing your work

Apache should still be stopped so start it using the *Apache Monitor*. Point your Web browser to *http://svn.yourcompany.com:81/svn* and you should see something similar to *Figure 24* above. If you do, congratulations, you've just integrated Subversion and Apache! When the Web request reaches Apache with the */svn* directory on the end of the URL, it knows – via the *Location* directive we added – to pass control to the Subversion/DAV server. This allows Apache to show all of the repositories in the *E:\svnrepos* directory. Since the directory is empty right now, nothing shows up. At this point we could leave things the way they are and move on to creating our first repository. But I'm not too happy with the URL used to access the repository. Since the Apache Web server is already configured with a sub-domain (*svn.yourcompany.com*) there's really no need to tack on the */svn* directory. We can easily nix this portion of the URL by using Apache Virtual Hosts. Virtual Hosts allow Apache to serve content from multiple domains. You could have *yourcompany.com* serve your normal Web site and *svn.yourcompany.com* serve your repositories. This provides a nice level of flexibility for your Web environment. With that in mind, let's reconfigure our repository access using Virtual Hosts instead of the *Location* directive.

## Enabling Apache Virtual Hosts

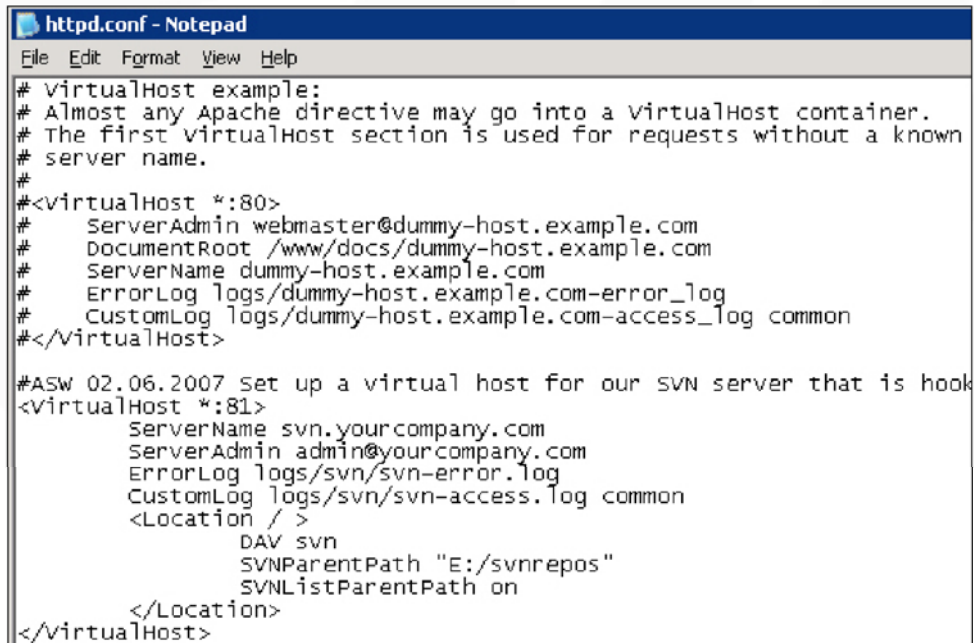


```
httpd.conf - Notepad
File Edit Format View Help
# Please see the documentation at
# <URL:http://httpd.apache.org/docs/2.0/vhosts/>
# for further details before you try to setup virtual hosts.
#
# You may use the command line option '-s' to verify your virtual host
# configuration.
#
# Use name-based virtual hosting.
#
NameVirtualHost *:81
#
# virtualHost example:
# Almost any Apache directive may go into a virtualHost container.
# The first virtualHost section is used for requests without a known
# server name.
#
```

Figure 25: Enabling Apache Virtual Hosts

Before you are able to utilize Virtual Hosts you must enable them in the *httpd.conf* file. Using the *Apache Monitor* stop Apache and find the *#NameVirtualHost* line in *httpd.conf*. To enable Virtual Hosts, remove the pound sign at the start of the line. Since my Apache Web server is listening on port 81, I also had to make sure my Virtual Hosts were configured for port 81 (\*:81 instead of \*:80). We're now ready to create our Virtual Host.

## Configuring a Virtual Host for Subversion



```
httpd.conf - Notepad
File Edit Format View Help
# virtualHost example:
# Almost any Apache directive may go into a virtualHost container.
# The first virtualHost section is used for requests without a known
# server name.
#
#<virtualHost *:80>
#   ServerAdmin webmaster@dummy-host.example.com
#   DocumentRoot /www/docs/dummy-host.example.com
#   ServerName dummy-host.example.com
#   ErrorLog logs/dummy-host.example.com-error_log
#   CustomLog logs/dummy-host.example.com-access_log common
#</virtualHost>
#ASW 02.06.2007 set up a virtual host for our svn server that is hook
<virtualHost *:81>
    ServerName svn.yourcompany.com
    ServerAdmin admin@yourcompany.com
    ErrorLog logs/svn/svn-error.log
    CustomLog logs/svn/svn-access.log common
    <Location />
        DAV svn
        SVNParentPath "E:/svnrepos"
        SVNListParentPath on
    </Location>
</virtualHost>
```

Figure 26: Creating the SVN Virtual Host

A Virtual Host is defined by a starting and ending *<VirtualHost>* tag. Within the body of the *VirtualHost* tag are directives that determine how the Virtual Host behaves. You probably noticed the *Location* directive which, other than one small change, is exactly the same as our previous version (Figure 23). The difference here is the

/ URL mapping in place of the previous /svn. What I'm doing here is pointing the entire domain to the Subversion repository instead of one directory (/svn). If your setup does not involve an *svn.yourcompany.com* sub-domain you may want to keep the /svn mapping. Other than the *Location* directive, the first important setting is \*:81 in the starting tag which says all traffic served by Apache should be directed through this Virtual Host. This means any and all domains that point to this server (and have port 81 in the URL) will obey the rules set forth in this Virtual Host container. The *ServerName* and *ServerAdmin* settings should be self-explanatory. The *ErrorLog* and *CustomLog* directives instruct Apache to store any Web logs for this Virtual Host in a custom location that we've nominated. This will allow us to separate any other Web server logs (like normal site logs) from our Subversion logs. Before this separation can occur we must create the appropriate log directory.

### *Creating the Custom Logs Directory*

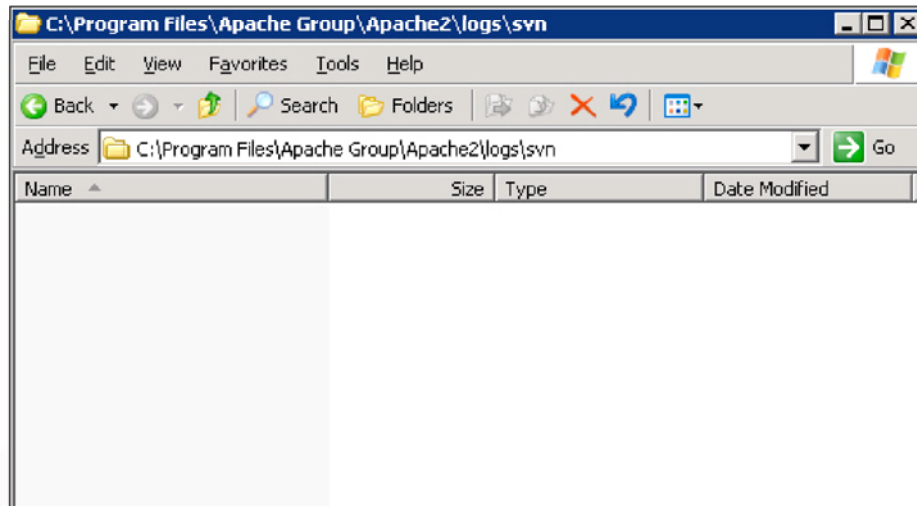


Figure 27: Create the svn log directory.

Create the *svn* directory in the *C:\Program Files\Apache Group\Apache2\logs* directory.



## Testing the Final Subversion – Apache Hook

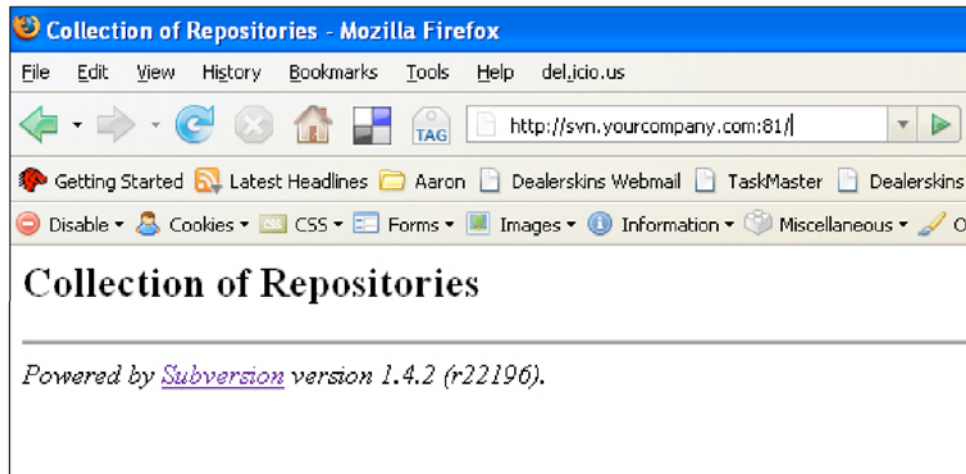


Figure 28: Testing the Virtual Host Configuration

With the Virtual Host now in place it's time to test our work. Start Apache and access the repository URL ([svn.yourcompany.com](http://svn.yourcompany.com) or [yourcompany.com/svn](http://yourcompany.com/svn) depending on your configuration). What you see should be exactly the same as what is displayed in Figure 24. This time, we are accessing the repository through a Virtual Host instead of a vanilla *Location* directive. Our changes also mean we now have two log files stored in `C:\Program Files\Apache Group\Apache2\logs\svn`. The `svn-error.log` file holds any errors generated by users accessing the repository (either through a Web browser or a Subversion client) while the `svn-access.log` file shows all the successful actions taken on the repository. These include checking out repository files, committing files, and more. If you open the `svn-access` log file you will see some entries from your recent Web access. The information displayed is not very readable or useful yet.

### Summary

Let's review our work in this part. We began by downloading and installing Subversion. Next, we verified the Subversion installation in a Web browser and configured several Apache/Subversion shared objects. We then created our repository directory, exposed the directory to Apache via the *Location* directive and tested our work in a Web browser. In order to make the environment more flexible we enabled Virtual Hosts, configured a Virtual Host for our Subversion sub-domain - which allowed us to create a dedicated Subversion logs directory - and finally we tested our work again in a Web browser.

At this point we are able to serve up repositories anonymously and we're logging all repository access. What we don't have are any actual Subversion repositories. To learn how to set this up read [Part 3: Installing TortoiseSVN and Creating Your First Repository](#). For information on refining the Subversion - Apache hook adding more meaningful logging and authenticated repository access, read [Part 4: Subversion and Apache - Better Logging and Authenticated Access](#). Finally, to learn how to integrate Subversion access into Eclipse review [Part 5: Accessing Subversion Repositories With Subclipse](#).

## Part 3: Installing TortoiseSVN 1.4.3 and Creating Your First Repository

In Part 1 I walked through the installation of the Apache Web server. In Part 2 I covered the installation of Subversion and the integration of Subversion and Apache. This involved configuring an Apache Virtual Host to handle all the requests that come from the Subversion sub-domain (*svn.yourcompany.com:81*). We also configured dedicated logging for all Subversion HTTP requests through appropriate Virtual Hosts directives. Finally, we looked at browsing the Subversion repositories via a Web browser, but since we hadn't created any repositories, this task was pretty unexciting. In this section I'll discuss the installation of TortoiseSVN, a popular client-side Subversion tool. Through TortoiseSVN we'll be able to create our first repository and perform our first repository import.

### *What is TortoiseSVN?*

TortoiseSVN is an amazingly easy to use and highly popular Subversion client for the Windows platform. Through extending Windows Explorer, TortoiseSVN commands are available anywhere within Windows by simply right-clicking. TortoiseSVN supports all the current Subversion protocols which include `http://`, `https://`, `svn://`, `svn+ssh://`, `file:///`, `svn+XXX://`. Using TortoiseSVN you can create, check out, update, and commit changes to repositories. You can also generate repository reports and view Subversion log files. We'll be installing TortoiseSVN on the server (in order to create the sample repository) as well as the client machine (so we can create working copies of the sample code).

### *Downloading TortoiseSVN*

Download the TortoiseSVN installer by pointing your Web browser to <http://tortoisesvn.net/downloads>. At the time of this writing the current version was 1.4.3. On the downloads page select the *TortoiseSVN-1.4.3.8645-win32-svn-1.4.3.msi* file.

### *Installing TortoiseSVN*

Getting TortoiseSVN installed is just a matter of stepping through the installation screens. If you already have an older version of TortoiseSVN, you can install 1.4.3 on top of your old installation. The installer will handle uninstalling the appropriate files and adding any new ones. At the end of the installation you will be prompted to restart your server. Remember, at this time you should be installing TortoiseSVN on the same computer that is running your Subversion server. We'll install TortoiseSVN on the client (your machine) later.

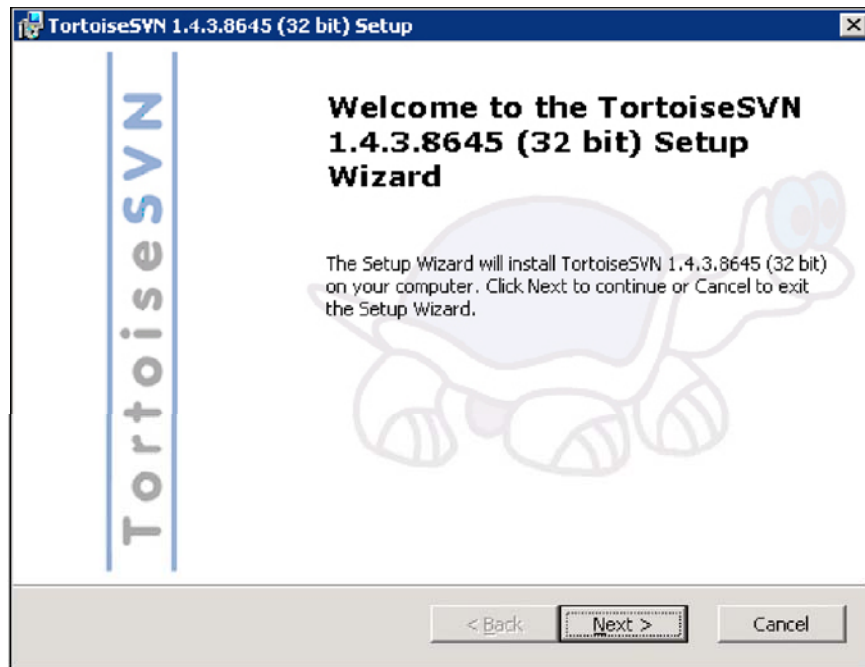


Figure 29: TortoiseSVN installation welcome screen

Locate the \*.msi file you just downloaded and double-click it to launch the installer. You'll be greeted by the TortoiseSVN welcome screen. Press *Next* to continue.

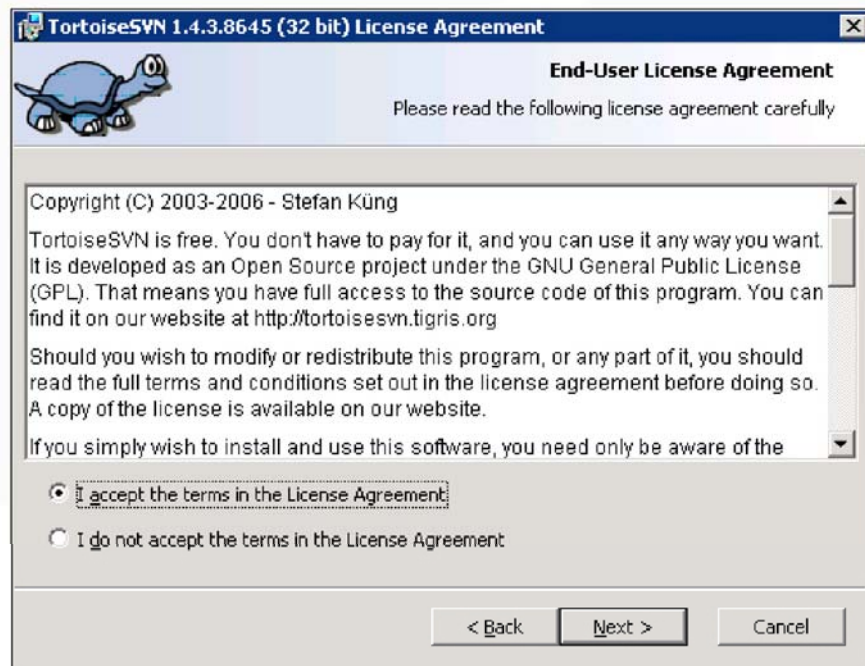


Figure 30: TortoiseSVN license agreement

Read the license agreement, accept the terms, and press *Next* to continue the installation.

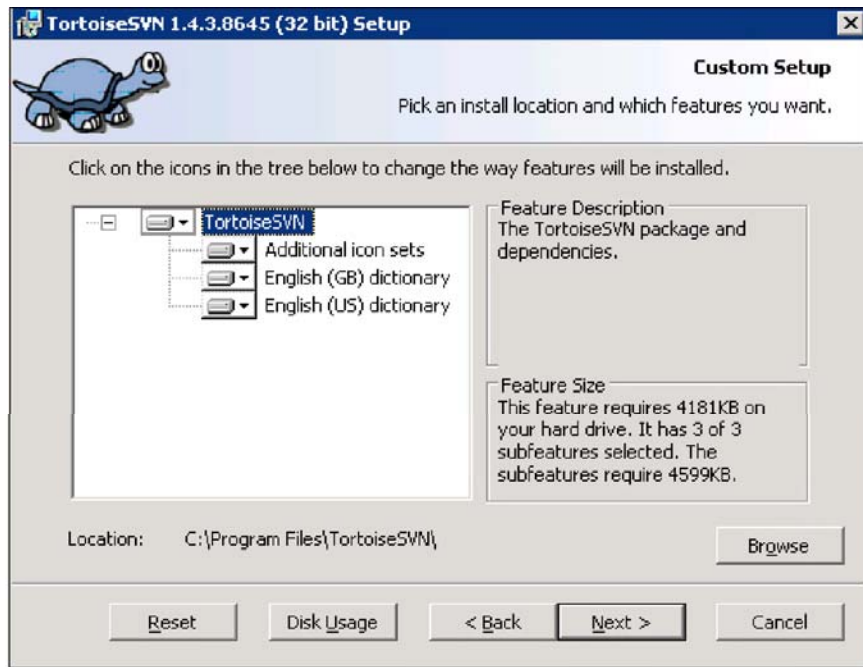


Figure 31: Custom setup screen

The installer defaults to placing TortoiseSVN in *C:\Program Files\TortoiseSVN\*, which is fine. Keep all the default settings and press *Next* to continue.

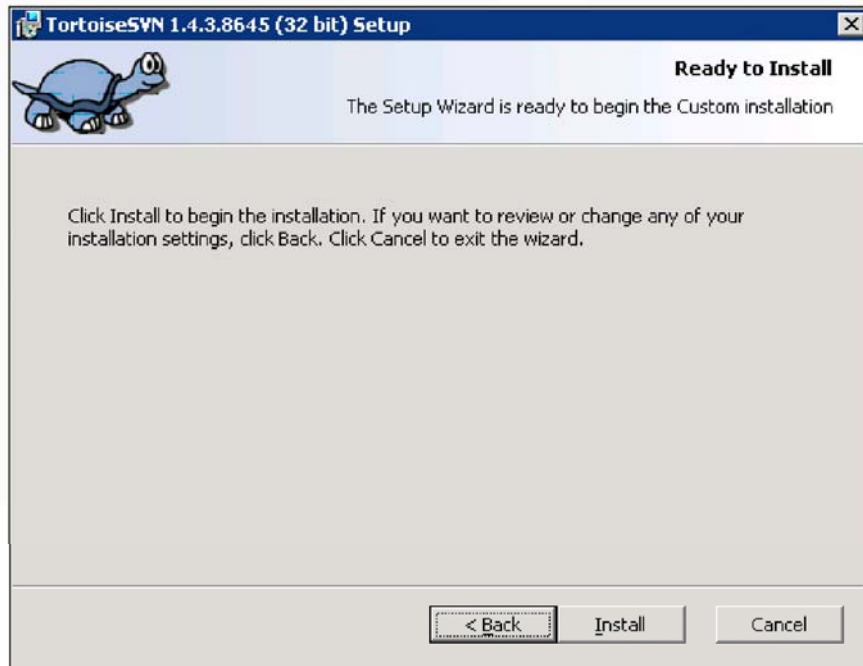


Figure 32: TortoiseSVN is ready to be installed

TortoiseSVN is now ready to be installed on the server. Press *Install* to start the installation.

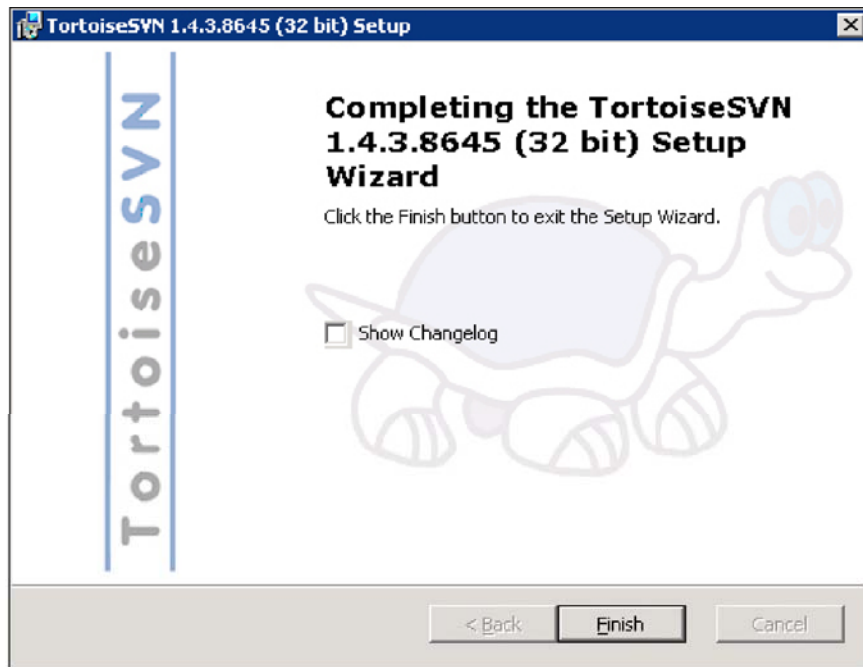


Figure 33: TortoiseSVN Installation Completion

After TortoiseSVN has been installed you'll see a completion screen. Press *Finish* to exit the installation.

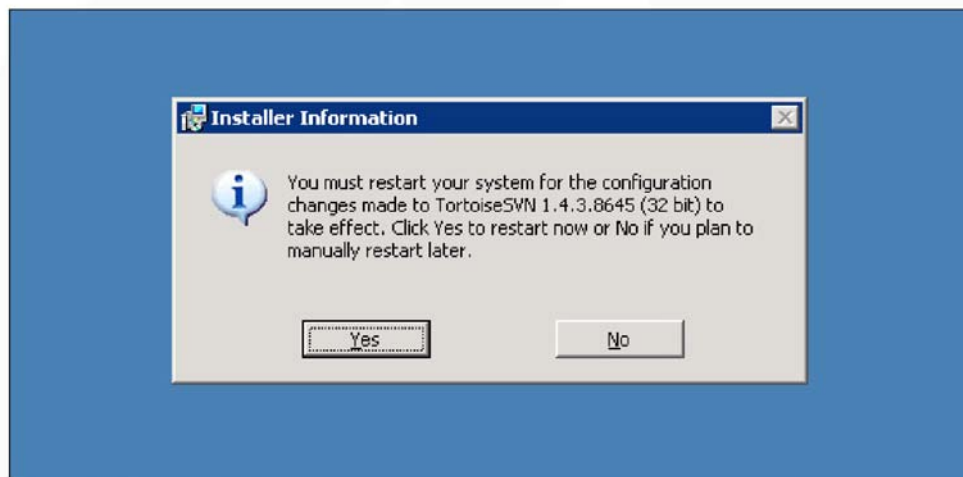


Figure 34: TortoiseSVN requires server restart

As I mentioned earlier, TortoiseSVN requires you to restart your computer in order for the changes to take affect. When your computer has finished restarting you'll have TortoiseSVN installed and operating.



## Creating Your First Repository

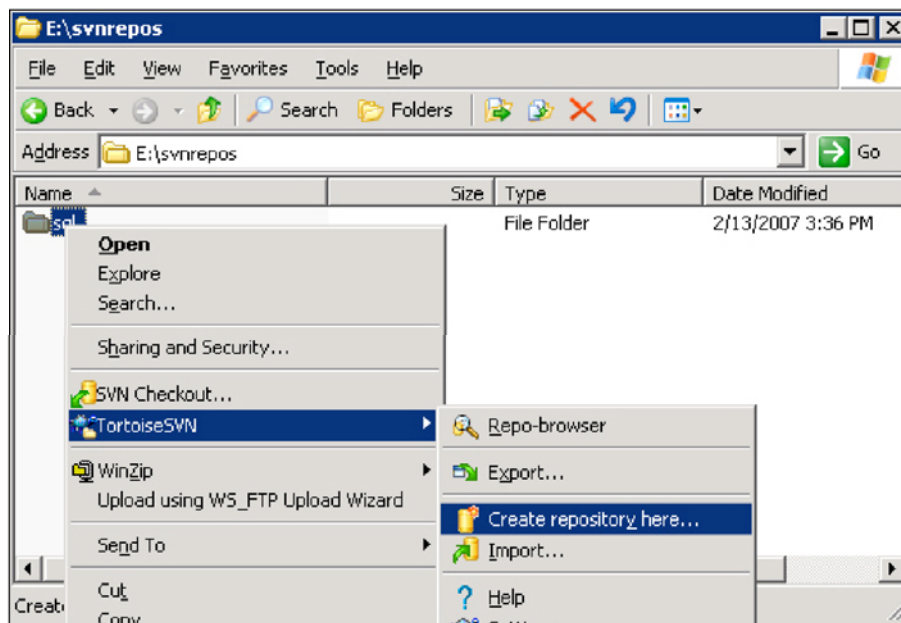


Figure 35: Creating your first repository

Remember the `svnrepos` directory we created in Part 2 (Figure 22)? It's been empty ever since as we finished preparing the Apache Virtual Host that points the `svn.yourcompany.com:81` domain to `E:\svnrepos`. You might also recall I steered you away from ever manually editing the contents of `svnrepos` as it's meant to be managed by the Subversion server. The one exception to this rule is the actions required to create the initial repository. To accomplish this, first create a new folder called `sql`. As an example we're going to set up versioning on a few really important SQL files. After creating the directory, right-click it and access the TortoiseSVN options in the right-click menu. Select the *Create repository here* option.

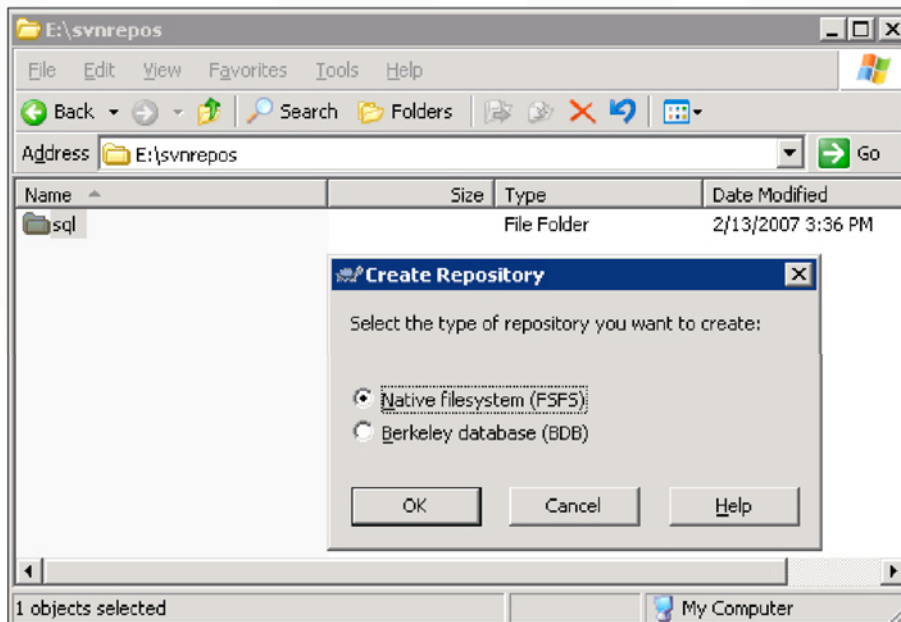


Figure 36: Selecting the filesystem type

Every Subversion repository is created using one filesystem type or another (how data is stored in the repository). Berkeley database (BDB) files are the original filesystem type having been around for quite a long time. The Native filesystem (FSFS) is a relatively new type but comes with a lot of benefits not found in BDB. It's outside the scope of this text to compare and contrast each type; if you're interested in learning more I recommend reading this Web page (<http://svn.collab.net/repos/svn/trunk/notes/fsfs>). The important thing to understand is Subversion manages the entire historical makeup of each repository using a flat file system. Personally, I recommend setting up all Subversion repositories using the Native filesystem (FSFS). Select the FSFS option and press Ok.

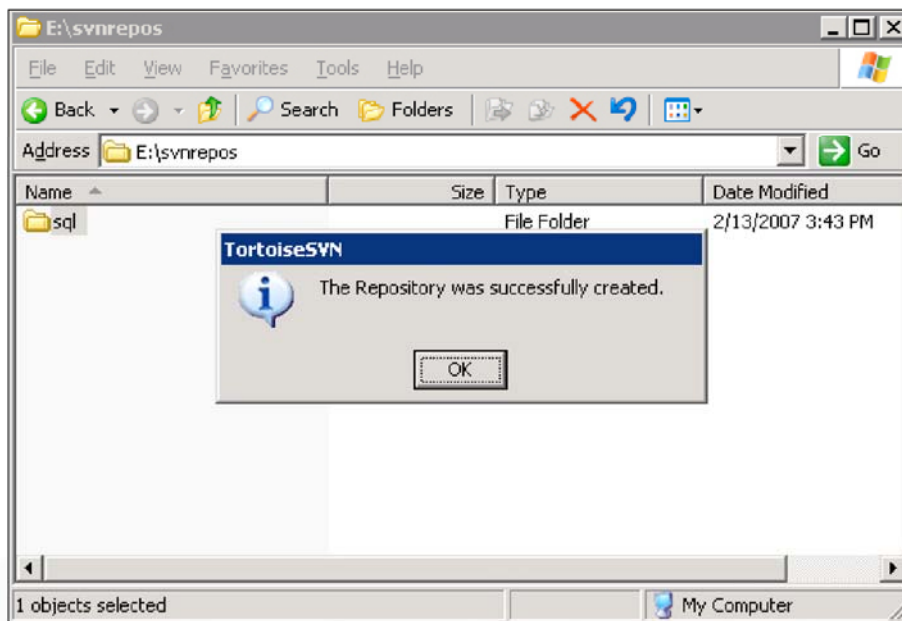


Figure 37: The new repository has been created

After a second or two TortoiseSVN will display a success message indicating the repository has been created. Behind the scenes, several directories and files were added to the *sql* folder that are needed to manage the new repository. However, our repository, for all accounts and purposes, is still empty. Before we change this by importing our sample SQL code, we need to stop and think about how to organize our data. There are several organizational structures common in Subversion. The most popular involves creating a few top-level directories: *trunk*, *branches*, and *tags* but what does these mean? The *trunk* directory is the main line of code for your development team. This is akin to the root folder of one of your Web sites or projects. This directory is also the most fluid receiving frequent bug fixes, updates, and revisions as your code evolves over time. The *branches* directory allows developers to branch-off or split the main line of development (the *trunk*). For example, let's say your team has written a really useful application for the marketing department. After 6 months of use marketing decides they need some relatively significant changes, enough to warrant a new release of the application. However, there are still a few outstanding bugs with version 1. Which project does your team, and more importantly each member of your team, focus? With branches, two senior developers can split version 1 from the trunk and start writing version 2. This development occurs without affecting the trunk whatsoever. In fact, the junior developer can focus on the version 1 bugs, at the same time, committing fixed code to the trunk and not negatively impacting version 2. When the senior developers have finished version 2 they can merge their work into the trunk picking up the bug fixes from the junior developer. Branches, whether they are configured as a *branches* top-level folder or not, are one of the greatest benefits to Subversions *copy-modify-merge* model. Almost as useful as *branches*, *tags* are snapshots of the repository at a particular point in time. The most common use of tags is creating releases or builds of your project. You can view tags as another synonym for milestones; when your team reaches a particular milestone it may be beneficial to take a snapshot of the trunk and give it a meaningful name like *release-1.0.3* or *version-1.5*.

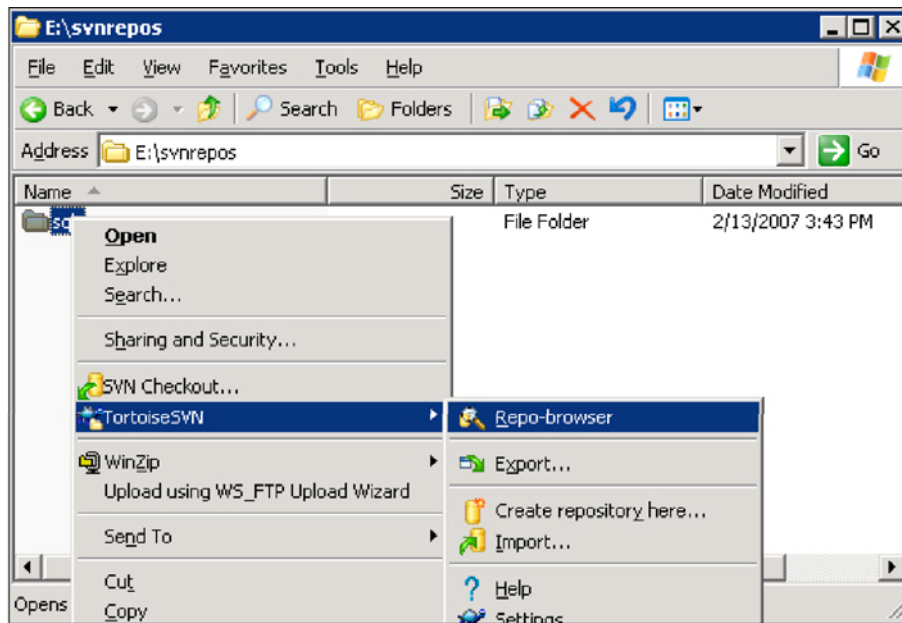


Figure 38: Opening the Repository Browser

Now that you understand a little more about the *trunk*, *branches*, and *tags* directories we need to add them to the *sql* repository. The easiest way to do this is to use TortoiseSVN. Using Figure 38 as an example, open the *Repository Browser* using the TortoiseSVN right-click menu.

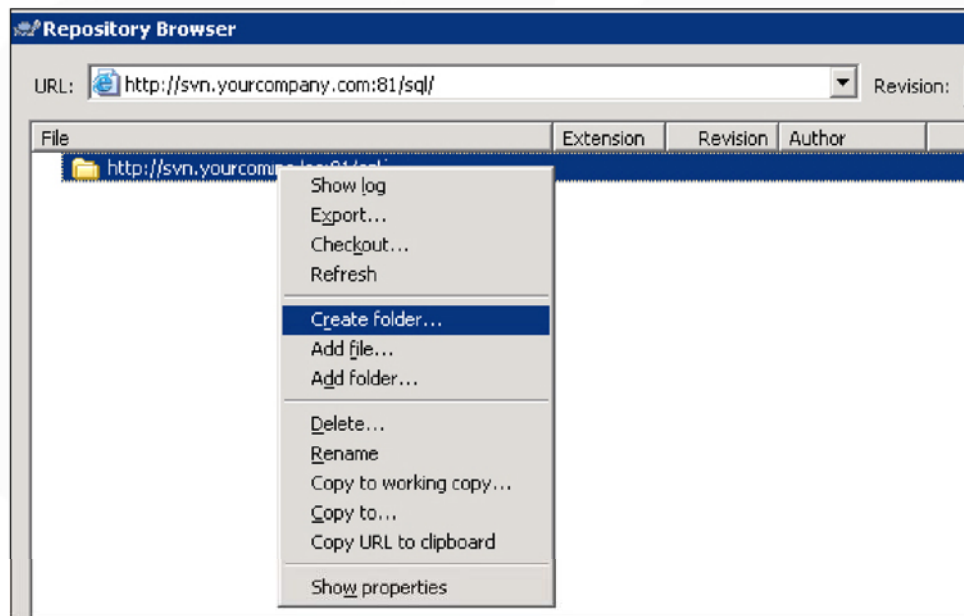


Figure 39: The Repository Browser

The *Repository Browser* (Figure 39) is a nifty tool allowing you to perform many actions on your repository. Some pertain to managing the repository while others enable simple methods of reporting repository activity. The first step to browsing the *sql* repository is to edit the URL address so it reads *http://svn.yourcompany.com:81/sql/*. The URL you use may be slightly different depending on how you configured things in Part 2. Once you've entered in the correct URL you should see the full HTTP path to the *sql* folder listed in the lower pane. Since the repository is empty, you cannot expand it to show its contents. We're now ready to create the *trunk*, *branches*, and *tags* directory. Using Figure 39 as an example right-click the



sql/ folder and select *Create folder*.

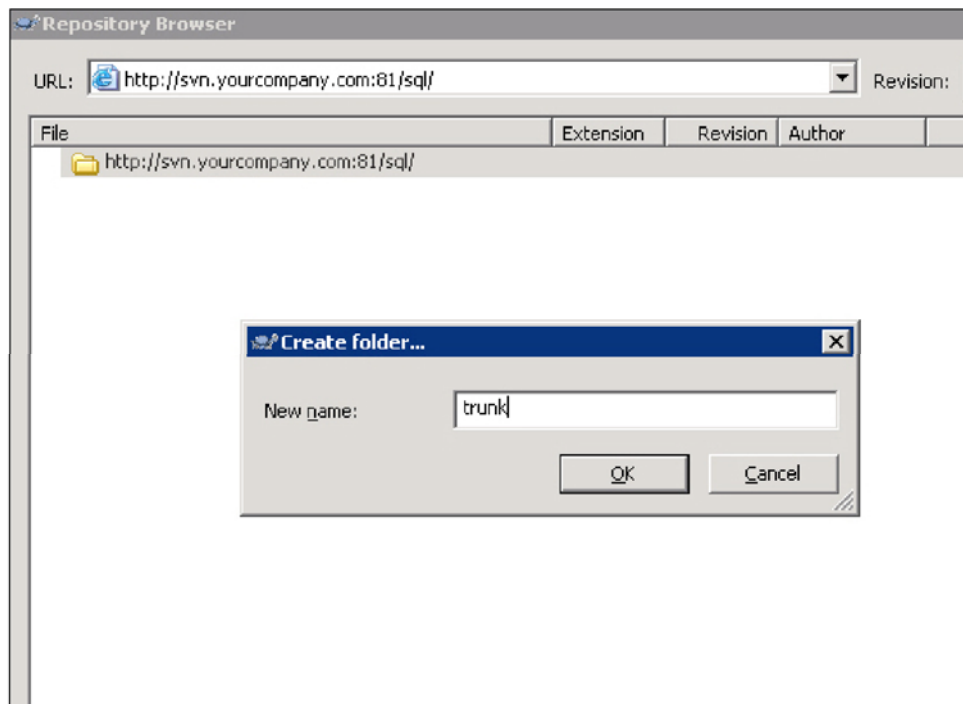


Figure 40: Creating the top-level trunk folder

Enter the folder name (trunk) in the window provided.

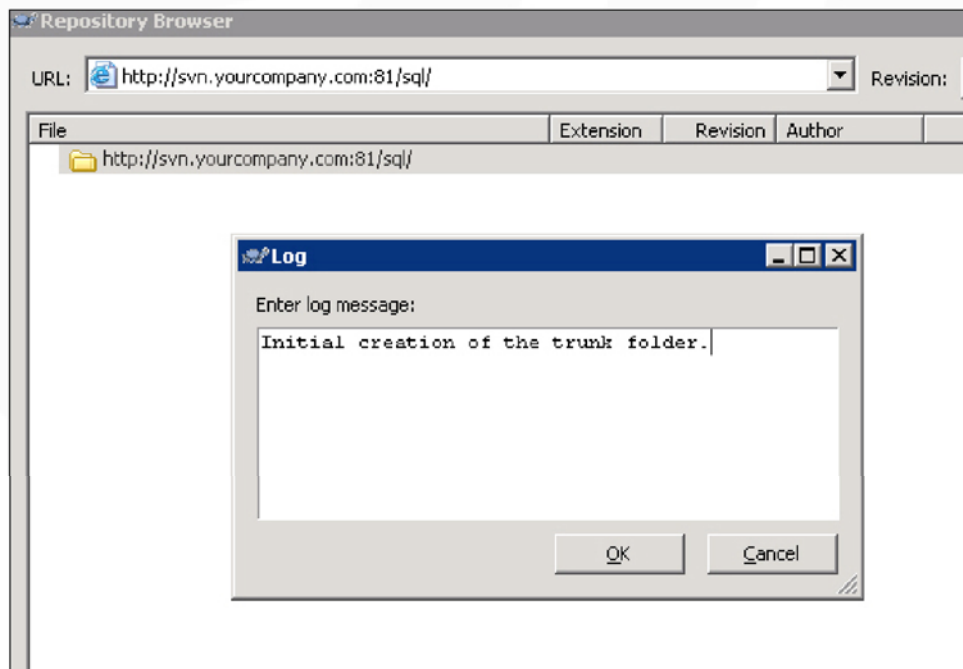


Figure 41: Entering comments for the trunk folder creation

Each time you make a change to the repository you are prompted to enter a log message pertaining to what you are adding, changing, deleting, etc. Enter an appropriate log message (Figure 41) and press *Ok*.

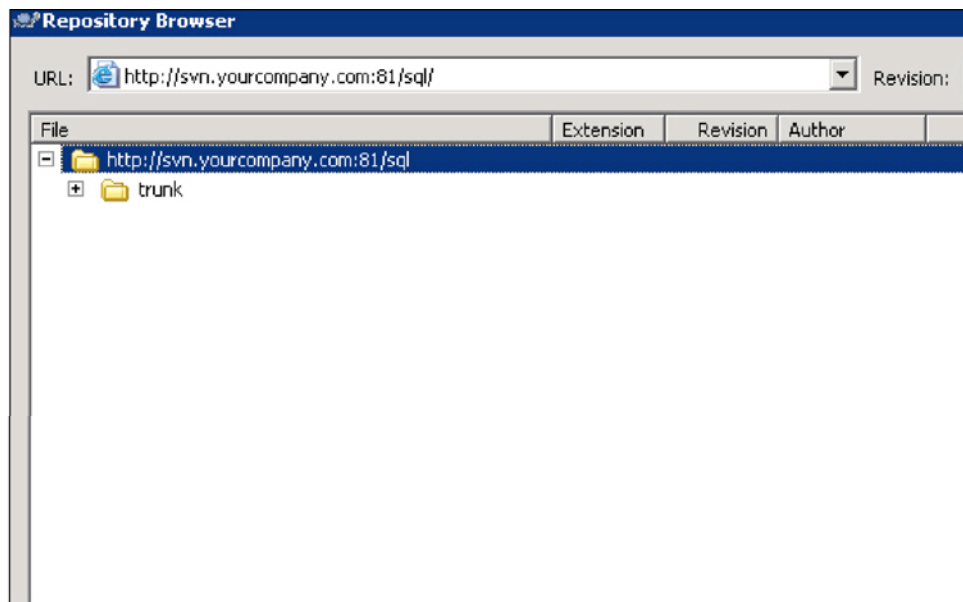


Figure 42: The trunk folder is now visible in the repository

The *sql* repository now has the *trunk* folder listed. To create the *branches* and *tags* directories repeat the above steps changing the log message for each folder you add. When you are finished, your repository should look similar to Figure 43 below.

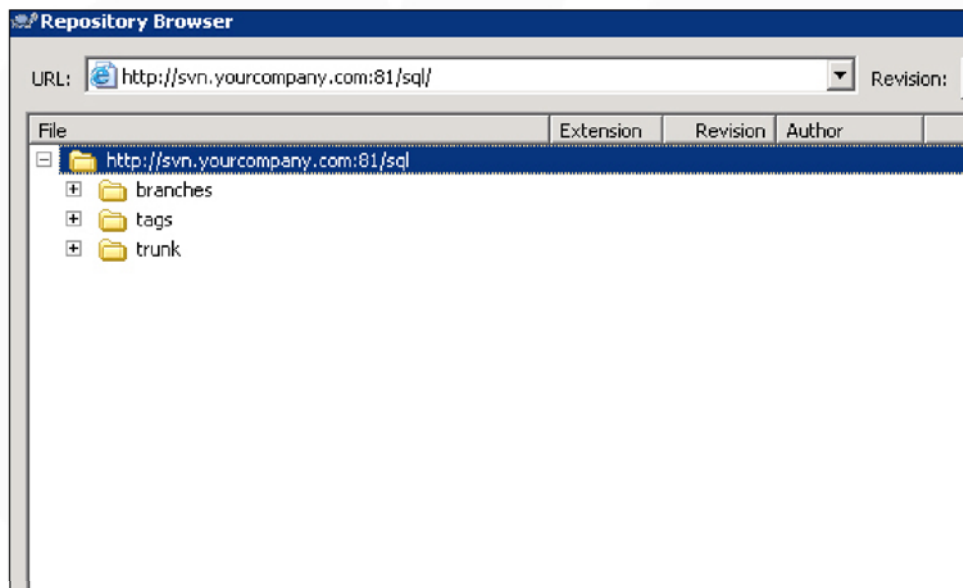


Figure 43: Final top-level folder configuration

With the top-level folders *trunk*, *branches*, and *tags* created we finally have our sample *sql* repository configured correctly.

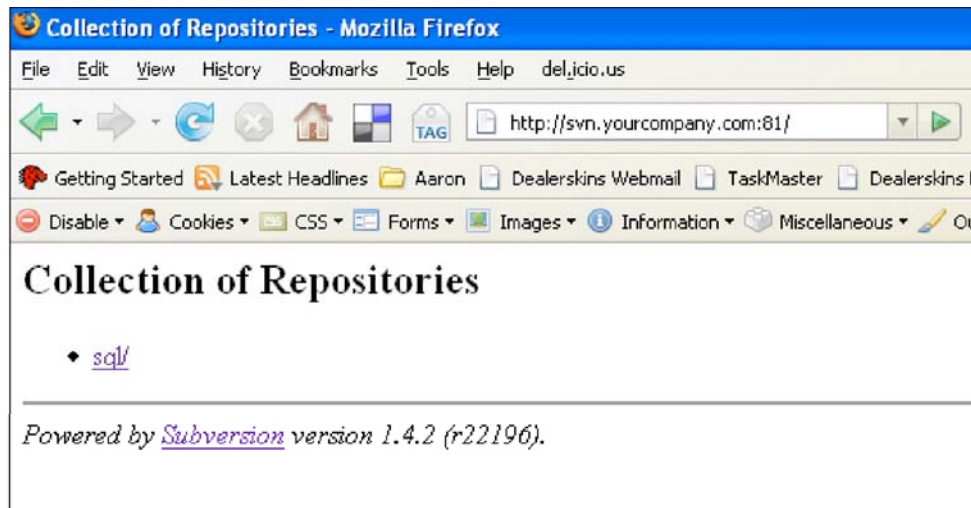


Figure 44: Checking the repository with a Web browser

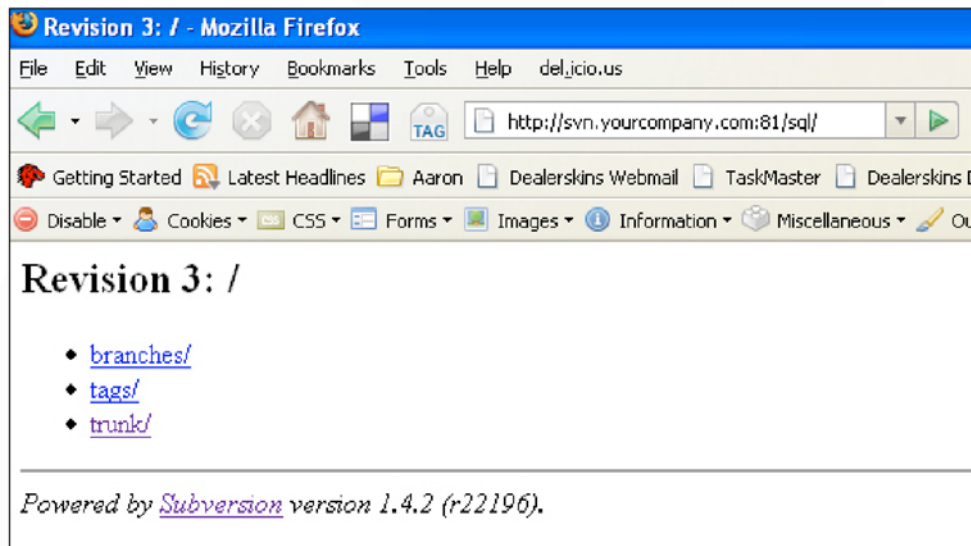


Figure 45: Drilling down into the repository

Before we move on let's check our repository using a Web browser. You should see a screen similar to *Figure 44* which lists all the repositories in the `E:\svnrepos` directory. Using the links provided you can drill-down into the `sql` repository (*Figure 45*) viewing any versioned files and directories. Notice how the repository is at Revision 3. This is because any change made, whether it's the addition of a directory or a change to an existing file, increments the overall revision number.

### Download the Sample SQL Code

I've created some sample folders and files - a very basic "project" - to make it easy to add content to the `sql` repository. They are located in the same download as this paper, in the *SQL Files* folder. Or, you can download them separately from here (<http://www.trajiklyhip.com/downloads/download.cfm?token=46D4A473-CDB5-5CCF-EB5E100FA1187318>).

## Import Sample SQL Code to the Trunk

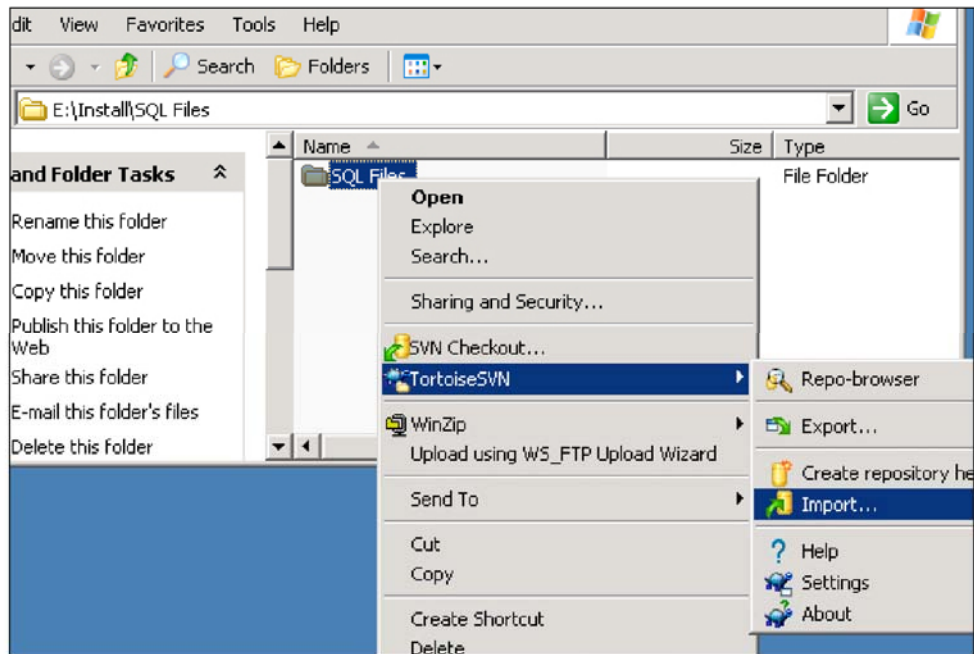


Figure 46: Importing code to the repository

Extract the SQL files from either zip to a temporary folder, being sure to retain the directory structure. You should have a main folder called *SQL Files* with several sub-folders and files. We're going to import the *SQL Files* folder into the repository trunk. This will cause the contents of the *SQL Files* folder – not the folder itself – to be imported into the trunk. The directory structure contained within the *SQL Files* folder should be retained as the contents are imported. Right-click the *SQL Files* folder, access the TortoiseSVN menu and choose *Import*.

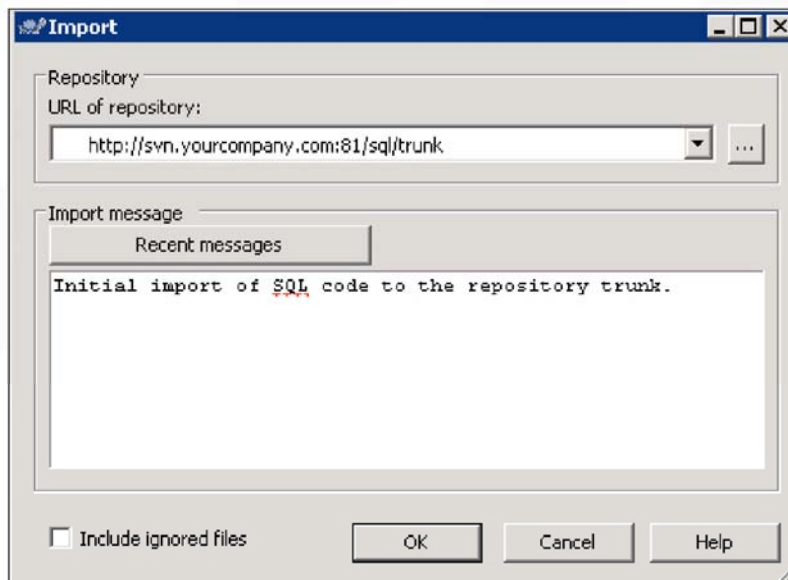


Figure 47: Selecting the sql trunk and entering a log message

To import the SQL files select the *trunk* folder of the *sql* repository. Enter an appropriate log message and press *Ok*.

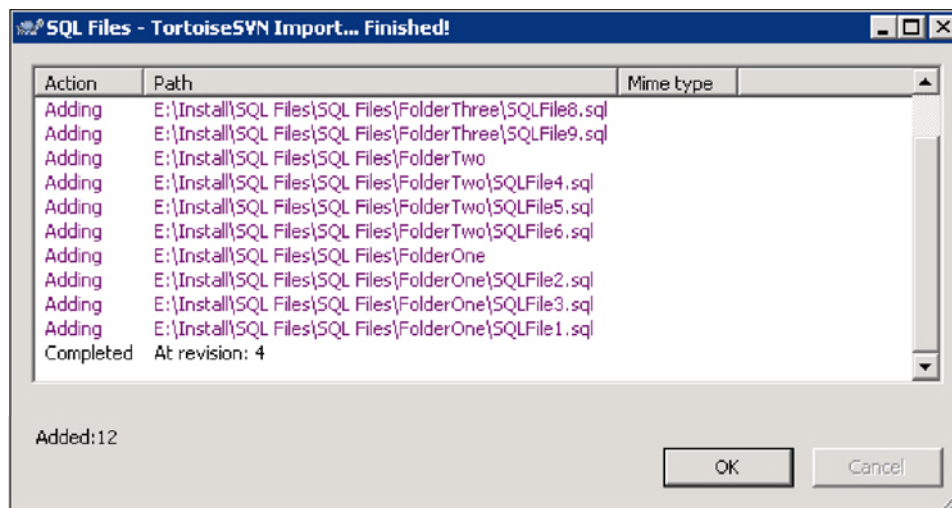


Figure 48: Import results

TortoiseSVN displays a log of each and every directory and file that was added to the trunk as well as indicating what the new revision number is. Press **Ok** to close the window.

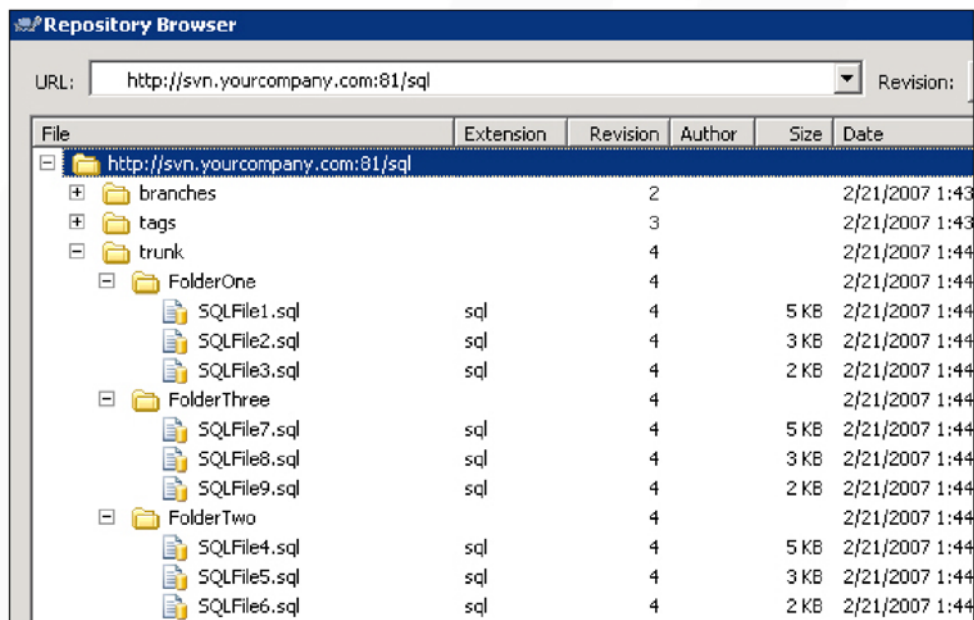


Figure 49: The trunk folder is now populated

To view the changes you just made open the Repository Browser and select the *sql/* repository. You may have to refresh the lower pane for the changes to show up.



Figure 50: Viewing the imported files in a Web browser

In addition to viewing the changes in TortoiseSVN's Repository Browser you can also view the changes in a Web browser. Drill-down into each folder and view a couple of the SQL files. If the files are basic text, like those I've provided, browsers like Firefox will display the contents of the file directly in the browser. Now that the trunk has versioned copies of files and directories, the repository can be checked out by any user who has a Subversion client (like TortoiseSVN) and access to <http://svn.yourcompany.com:81/sql/>.

### Creating a Working Copy of the Repository

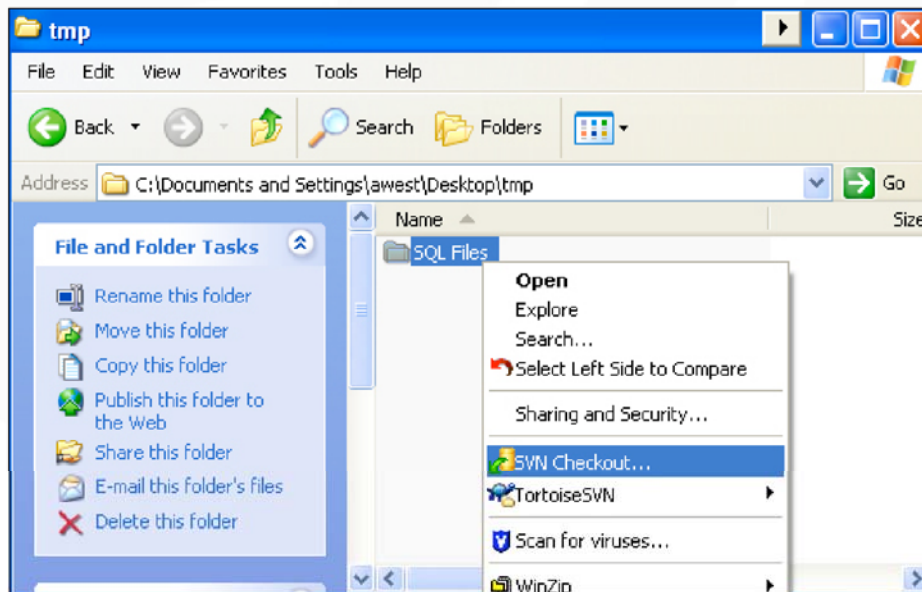


Figure 51: Checking out the repository

Up to this point all our work has been focused on the development server. We're now ready to move over to a client machine, install TortoiseSVN and create a working copy of the *sql* repository. Revisit the TortoiseSVN installation instructions if needed and return here once TortoiseSVN is installed on the client machine. The first step to setting up a working copy is to create a folder where the project will live. In Figure 51 above, I've created a folder called *SQL Files* on my desktop. Incidentally, this probably isn't a great place for the project



but will work fine for demo purposes. With the *SQL Files* folder created, right-click the folder and select *SVN Checkout*.

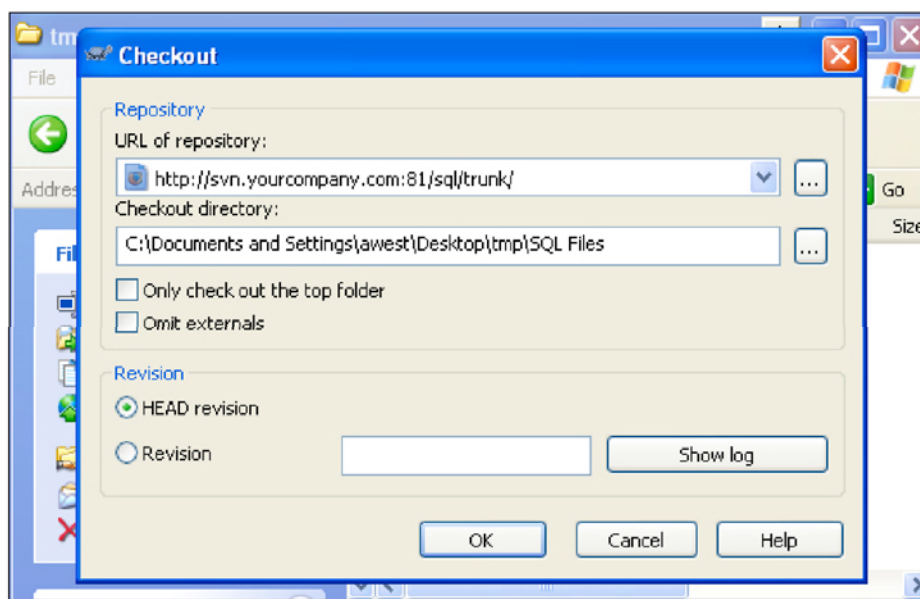


Figure 52: Selecting the repository URL and Checkout directory

TortoiseSVN prompts you to select the URL of the repository you want to checkout to the client machine. Be sure and select the *trunk* folder of the repository. The *Checkout directory* should already be pre-filled so all that's left is determining which revision we want. The *HEAD revision* refers to the latest and greatest contents of the repository. If for some reason you wanted an earlier revision you can select the *Revision* radio button and type the revision number in the box provided. For our purposes leave the *HEAD revision* selected and press *Ok*.

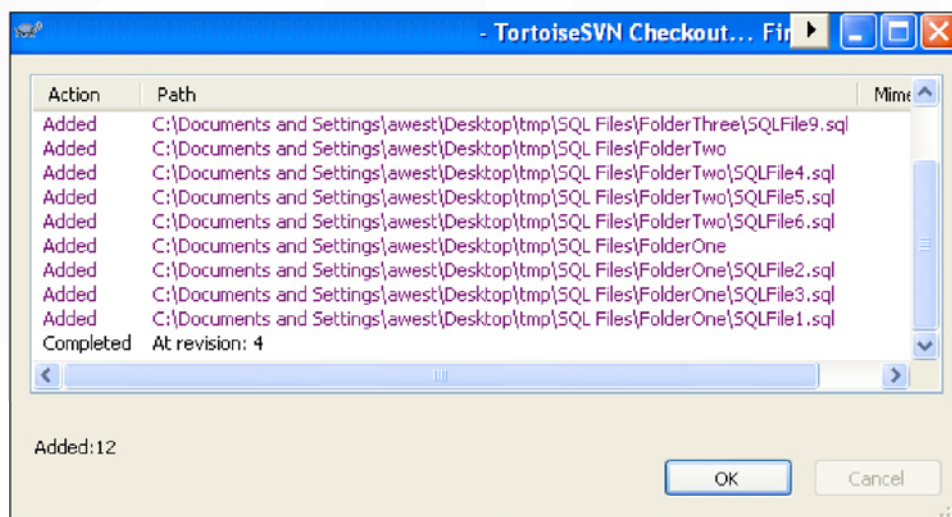


Figure 53: The working copy is populated with repository contents

TortoiseSVN will access the repository URL and download the entire contents of the *trunk* adding all files and folders to the local working copy. With this in place, you can now work on the files on your local computer making any changes you need. Changes you make on your local computer are not reflected in the repository on the Subversion server until you commit them. Let's take a look at this process.

## Committing Changes to the Repository

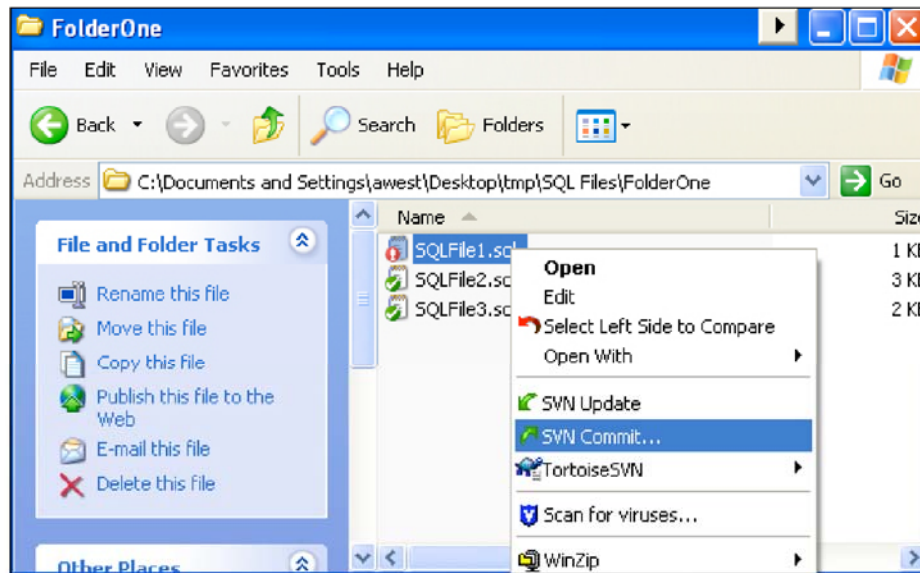


Figure 54: Committing a change to the repository

Navigate to the *FolderOne* folder of your working copy and open up *SQLFile1.sql*. Make any change you want to the file and save it. Return to the *FolderOne* folder and press *F5* to refresh Windows Explorer. The icon for the changed file should now look a bit different than the others. The exclamation point icon indicates you have a file changed in your working copy that has not been committed to the repository. This is extremely handy as you make dozens of changes across many files in a project. Using *Figure 54* as an example, right-click the changed file and select *SVN Commit*.

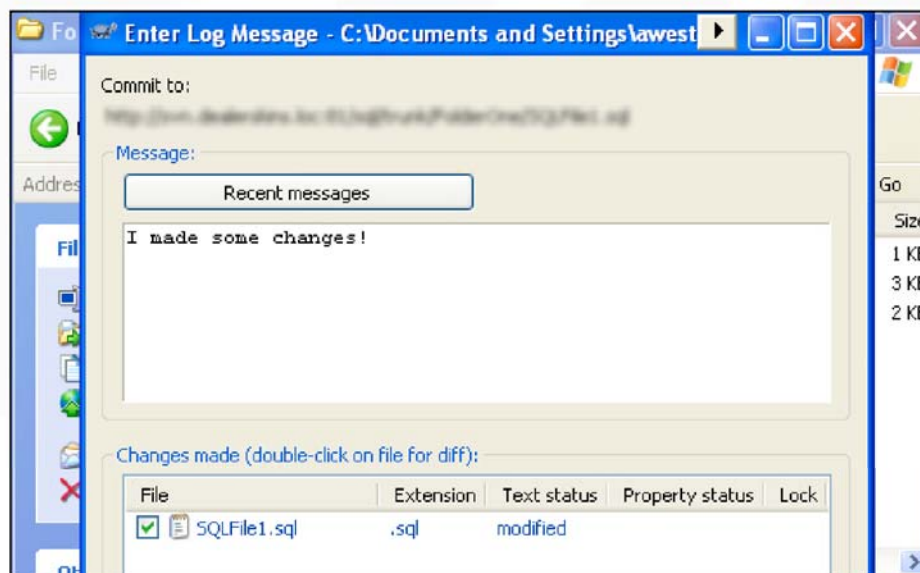


Figure 55: Selecting which file(s) to commit and adding a log message

As always, TortoiseSVN prompts you to enter a log message about what you have changed. The lower section of the window will list any local files that have changed. This gives you the ability to be selective about what you want to commit.



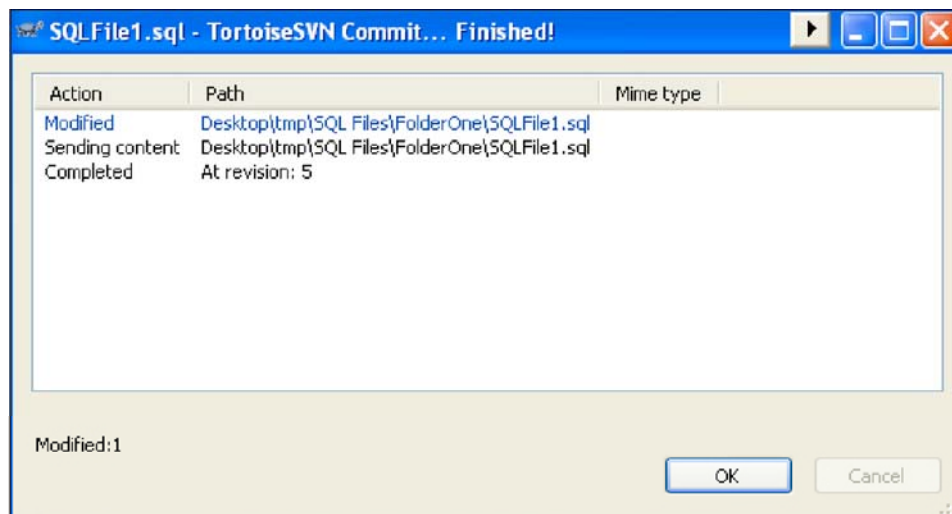


Figure 56: Commit results

Once the files have been committed to the remote repository, TortoiseSVN shows you a status message indicating the new revision number.

### Getting Changes from the Repository

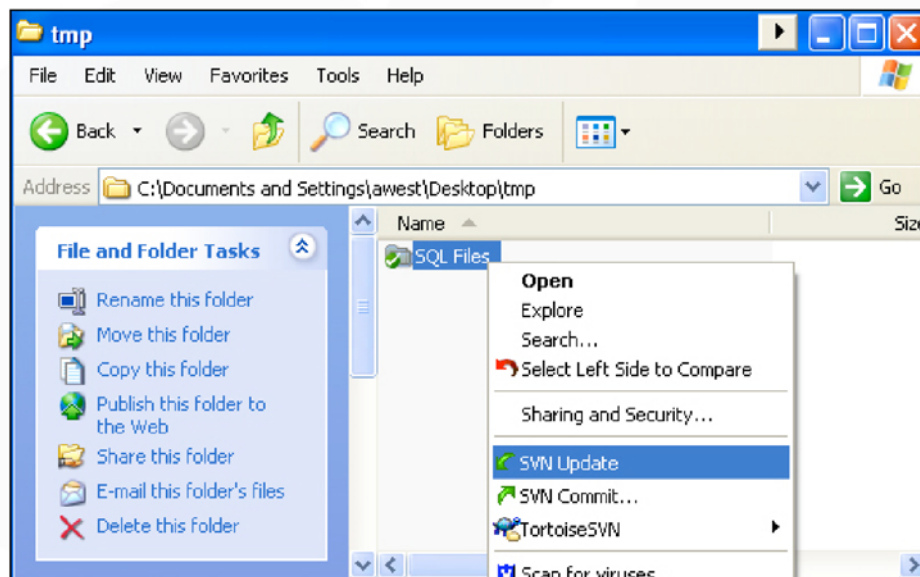


Figure 57: Getting changes from the repository

The last piece to the round-trip cycle of revision control is retrieving other developers changes from the repository. This is known as *updating* your local working copy. To do this with TortoiseSVN you select your local project folder – in this case *SQL Files* – right-click the folder and select the *SVN Update* option. TortoiseSVN will poll the remote Subversion server and compare your local copy to that of the repository. If the repository has changes you don't, TortoiseSVN will download those changes and display an appropriate status message.

## Summary

In this part of the text I covered key aspects to creating and managing a sample Subversion repository. First, we installed the TortoiseSVN Subversion client on the server and used it to create our sample repository. Along the way I discussed a common Subversion repository strategy – creating the trunk, branches, and tags top-level folders. Then, I walked through the steps to importing sample SQL code into the repository trunk. And finally, I covered working copy creation, committing changes to the repository, and requesting updates from the repository.

In Part 2, I covered the integration of Subversion and Apache which included creating custom logs of all Subversion repository access. To learn how to further refine the Subversion – Apache hook, adding more meaningful logging and authenticated repository access, check out [Part 4: Subversion and Apache – Better Logging and Authenticated Access](#). To learn more about accessing Subversion repositories from the client-side perspective, check out [Part 5: Accessing Subversion Repositories With Subclipse](#).

## Part 4: Subversion and Apache – Better Logging and Authenticated Access

At the end of Part 2 we briefly looked at Apache's logging of Subversion's "traffic." We'll revisit this topic in this section discovering a better way to configure logging. We'll also address repository security adding a couple DAV directives that create authenticated repository access. Let's get right to it.

### Default Apache Logging

```
#ASW 02.06.2007 Set up a virtual host for our SVN server that is hook
<VirtualHost *:81>
    ServerName svn.yourcompany.com
    ServerAdmin admin@yourcompany.com
    ErrorLog logs/svn/svn-error.log
    CustomLog logs/svn/svn-access.log common
    <Location / >
        DAV svn
        SVNParentPath "E:/svnrepos"
        SVNListParentPath on
    </Location>
</VirtualHost>
```

Figure 58: Default Apache Logging

In Part 2 we discussed the *ErrorLog* and *CustomLog* directives and I walked you through creating the *svn* folder that holds the Subversion logs. I did not however, discuss the log format. In *Figure 58* the *CustomLog* directive has a final piece to it that instructs Apache the format to use for the access log. The *common* format is the base format Apache uses in most cases. Log output for this format looks like the following:

```
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "MKACTIVITY /sql/svn/act/8db7ca3c-4a26-3243-8385-ff3ec9" 200 438
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/trunk/FolderOne HTTP/1.1" 207 388
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/svn/vcc/default HTTP/1.1" 207 388
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "CHECKOUT /sql/svn/bln/4 HTTP/1.1" 201 338
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPPATCH /sql/svn/wbl/8db7ca3c-4a26-3243-8385-ff3ec9" 200 687
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/trunk/FolderOne HTTP/1.1" 207 404
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "CHECKOUT /sql/svn/ver/4/trunk/FolderOne/SQLFile" 200 687
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PUT /sql/svn/wrk/8db7ca3c-4a26-3243-8385-ff3ec9" 200 687
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "MERGE /sql/trunk/FolderOne HTTP/1.1" 200 687
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "DELETE /sql/svn/act/8db7ca3c-4a26-3243-8385-ff3ec9" 200 687
```

Figure 59: Apache's Common Log Format

Other than listing the WebDAV actions (MKACTIVITY, PROPFIND, CHECKOUT) there's not a lot of useful stuff in here. Sure you can view the date and time actions were taken by various Subversion clients, but the target of those actions is harder to determine. Through a couple minor edits to our Virtual Host we can really clean up the *svn-access.log* file.

### Custom Apache Logging

```
#ASW 02.06.2007 Set up a virtual host for our SVN server that is hooked to Apache
<VirtualHost *:81>
    ServerName svn.yourcompany.com
    ServerAdmin admin@yourcompany.com
    ErrorLog logs/svn/svn-error.log
    CustomLog logs/svn/svn-access.log "%t %u %e" env=SVN-ACTION
    <Location / >
        DAV svn
        SVNParentPath "E:/svnrepos"
        SVNListParentPath on
    </Location>
</VirtualHost>
```

Figure 60: Better Apache Logging

Stop Apache using the Apache Monitor and open the *httpd.conf* file. Find the appropriate Virtual Host section

and delete the *common* format. In it's place, type (or copy/paste) the following: `"%t %u %{SVN-ACTION}e"` `env=SVN-ACTION`. This format includes several key variables that get replaced by runtime values. First, `%t` gets replaced by the current server date and time. `%u` gets replaced by the username of the person accessing the Subversion repository. The really important parts are the *SVN-ACTION* environment variables. These are set by the *mod\_dav\_svn* module whenever an action is taken by a Subversion client. Apache replaces this variable with the value set by *mod\_dav\_svn*. The result, is more readable and easily understandable access logs. To prove this, start Apache on the server and return to the client machine. Right-click your working copy of the SQL files and select the *SVN Update* TortoiseSVN option. Now, return to your server and open the *svn-access.log* file once again. The last line should look something like this:

```
[21/Feb/2007:17:19:00 -0600] - update '/trunk'
```

Obviously, this is a bit more meaningful. We know the action taken by the client was a Subversion *update*, and the target of the update was the clients entire *trunk* folder. What we don't know yet is *who* requested the update. In order to get the `%u` variable replaced by usernames we need to set up some basic Apache authentication and authorization.

### Basic Apache Authentication

Right now, your repository is anonymously available to anyone with Web access to it. If your Apache Web server is available to the general public this means anyone in the world can access the repository. Users can checkout working copies of the repository using their Subversion client of choice, they can browse the latest repository version via a Web browser, and they can commit changes. All anonymously. There are several ways to set up Apache authentication depending on the needs of you and your team. I'm going to describe how to set up basic HTTP authentication but I want to make something clear, this is not a super-secure, hard to hack security set up. If you need a really robust, encrypted authentication scheme, you need to read more about what is possible with Apache, HTTPS, and SSL. Basic HTTP authentication provides over-the-wire username/password challenges with the information sent in near clear text. This type of authentication is fine if you're an individual or a team member setting up Subversion within a protected company network. The basic steps are to generate an Apache password file that stores usernames and passwords of individuals who will access the repository. Then, direct the Virtual Host to require authentication for certain actions. To get started, I recommend creating a special directory for holding the authorization file.

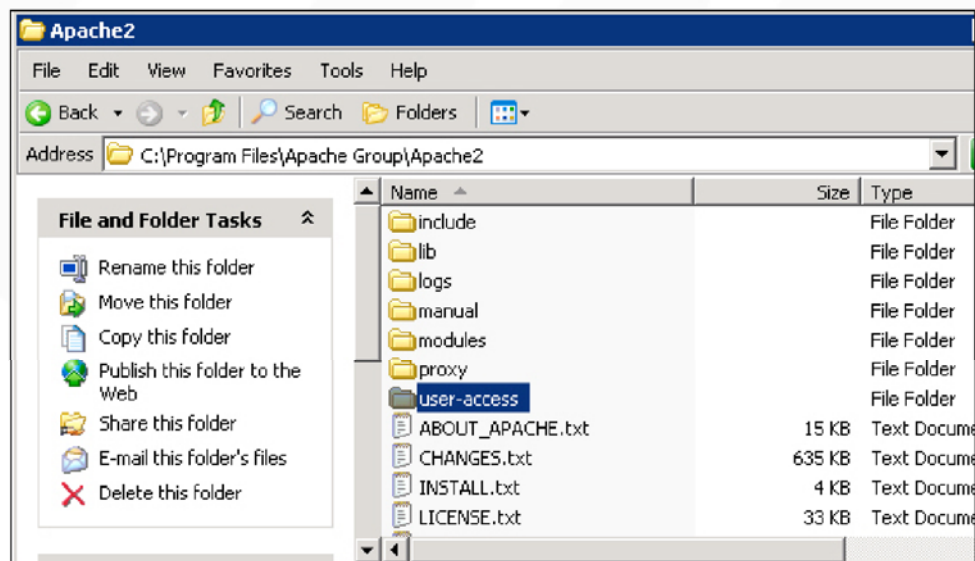


Figure 61: Creating the user-access folder

Create the *user-access* folder in Apache's root directory. To generate an authorization file we'll call on Apache's built-in *htpasswd* utility. Open a command prompt (*Start->Run->cmd*) and navigate to Apache's bin folder.

Inside the bin folder is the *htpasswd* utility. To create the password file and add the user *awest* at the same time issue the following command:

```
C:\Program Files\Apache Group\Apache2\bin>htpasswd -cm ../user-access/svn-auth-file awest
```

The first part of the line above is the directory the command is being run from. The command itself starts with nominating the *htpasswd* utility followed by the *-cm* switches. The *-c* switch informs the utility we are creating the authorization file for the first time. The *-m* switch tells *htpasswd* to use the MD5 encryption algorithm when storing the password in the file. Next, we tell *htpasswd* where the file will be stored (in the *user-access* directory – which is one directory above the *bin* directory) and what it will be called (*svn-auth-file*). Finally, we nominate the first username to store in the file. When you type this command and press enter you'll be prompted for a password and password confirmation. *htpasswd* will then generate the *svn-auth-file* for you. If you only need to add one username and password combination you are done. However, if you are working on a team you will want to add the members of your team to the password file. You do this by issuing a similar command.

```
C:\Program Files\Apache Group\Apache2\bin>htpasswd -m ../user-access/svn-auth-file jdoe
```

The only difference here is the absence of the *-c* switch. It's not needed since we've already created the password file. Now that we have a basic authorization file we need to make changes in the Virtual Host that requires authorization. If Apache is running, stop it with the *Apache Monitor*.

```
#ASW 02.06.2007 Set up a virtual host for our svn server that is hook
<VirtualHost *:81>
    ServerName svn.yourcompany.com
    ServerAdmin admin@yourcompany.com
    ErrorLog logs/svn/svn-error.log
    CustomLog logs/svn/svn-access.log "%t %u %{SVN-ACTION}e" env=
    <Location / >
        DAV svn
        SVNParentPath "E:/svnrepos"
        SVNListParentPath on

        AuthType Basic
        AuthName "Company SVN"
        AuthUserFile user-access/svn-auth-file
        <LimitExcept GET PROPFIND OPTIONS REPORT>
            Require valid-user
        </LimitExcept>
    </Location>
</VirtualHost>
```

Figure 62: Adding Authentication Directives to *httpd.conf*

There are four basic parts to the authorization directives in *Figure 62*. First, we tell Apache what type of authorization (*AuthType*) we are setting up. Next, we set an authorization realm name (*AuthName*). This name will show up in various places when users are asked to authenticate. Then, we let Apache know where our authorization credentials are stored (*AuthUserFile*). This is the password file we created a few moments ago. Finally, we create a *Limit* directive that instructs Apache when to require authorization. In the example in *Figure 62* we are allowing anonymous access to basic read-only repository actions. These include checking out working copies and browsing the repository via a Web browser. Actions like committing changes require a user to validate who they are against the password file. If you wanted to require authorization 100% of the time you could set this up by deleting the *Limit* start and end tag and leaving the *Require valid-user* statement. To test these new changes start Apache on the server. Then, from the client machine, right-click your working copy directory and select *SVN Update*. The update process occurs without incident and you aren't prompted to enter your username and password. This is because updates are considered read-only actions. Now, make a change to one of the SQL files and commit the change using the *SVN Commit* option. You should be prompted to enter your username and password. The challenge window will look something like this:



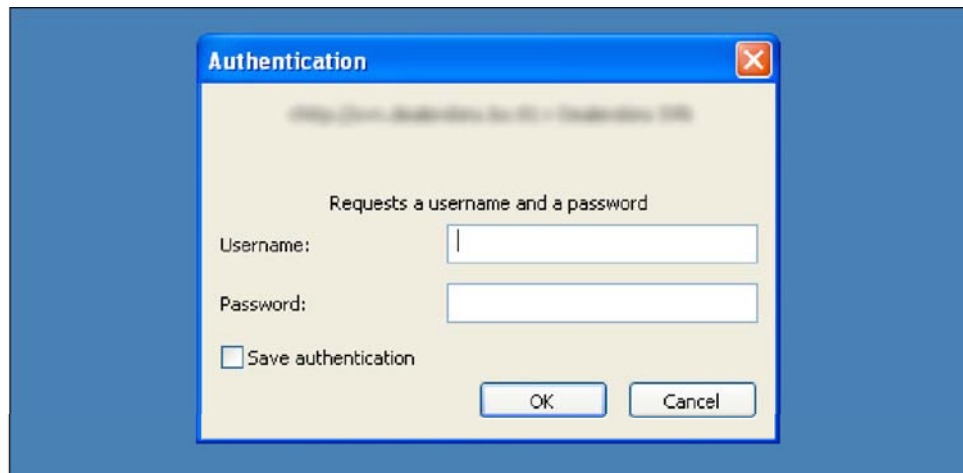


Figure 63: Authorization is required for commits.

Enter the username and password you set up and press *Ok*. You can click the *Save authentication* check box if you want TortoiseSVN to remember your credentials. If you enter a valid username and password combination your changes should be committed and the version of the repository should increase by 1. You can verify this using a browser and visiting your repository (<http://svn.yourcompany.com:81/sql/>) or, using the Repository Browser. Right-click your working copy directory and select TortoiseSVN's *Show Log* option.

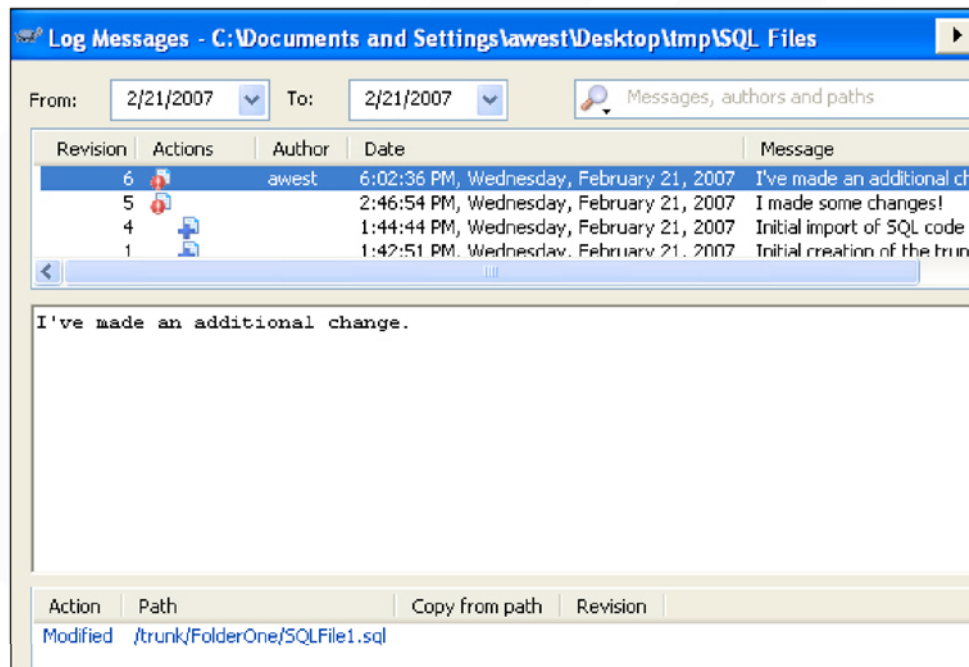


Figure 64: Usernames are listed in the Author column

Notice the username listed in the Author column in *Figure 64*. To bring things full circle, open the *svn-access.log* file on your Apache server. The last two lines should look very similar to the lines below. Any Subversion action that requires authentication will show in the log file with the action that was taken and who performed it.

```
[21/Feb/2007:18:02:36 -0600] awest commit r6
[21/Feb/2007:18:04:14 -0600] - log-all '/trunk
```

## Summary

In this part I walked through the steps necessary for creating more meaningful Apache/Subversion logging and setting up basic authenticated repository access for the actions you probably care about most (i.e. the most destructive ones). There are tons more you can do in terms of authentication and authorization, including locking things down more tightly and employing a more complicated authorization system than basic HTTP authorization. For instance, you can integrate Apache with your LDAP server (such as Active Directory) enabling clients to authenticate using their Windows username and password. You can also take the password file to another level setting up path-based authorization. For instance, Jane should have read access to certain parts of the *sql* repository while having write access to others. There are a lot of resources available that can help setting up either of these scenarios. For the latter, check out the *mod\_authz\_svn* module documentation. One word of warning: while it's not overly complicated to set up this type of environment, it may not be needed and does have a maintenance cost associated with it. Remember, Subversion is a source control server, if someone commits something they shouldn't the change can always be undone!



## Part 5: Accessing Subversion Repositories With Subclipse

Throughout the parts of this text I've used TortoiseSVN to perform all repository actions. This was mainly due to the simplicity and ease of use TortoiseSVN affords. TortoiseSVN is also handy when versioning assets that aren't code-related like spreadsheets and general documents. When working with code - ColdFusion for instance - there are other Subversion clients that work just as good as TortoiseSVN and don't require you to leave your development environment to request repository updates or commit changes. One such tool is Subclipse which is built on the open-source Eclipse platform. Eclipse is an extremely popular Java-based programming tool that works with just about every modern programming language. I use Eclipse and Eclipse plugins like CFEclipse on a daily basis to manage my code. In this section, I discuss installing, configuring, and using Subclipse, the Subversion plugin for Eclipse.

### *What is Subclipse?*

Just like TortoiseSVN, [Subclipse](#) is a Subversion client tool allowing you to create and manage working copies of versioned resources. The difference between the two is Subclipse is designed around the open-source Eclipse Platform. In order to manage and work with repositories using Subclipse, you must first download and install Eclipse. The rest of this section assumes you have Eclipse 3.2 installed and operating properly and walks you through installing Subclipse and using it to manage Subversion repositories.

### *Installing Subclipse*

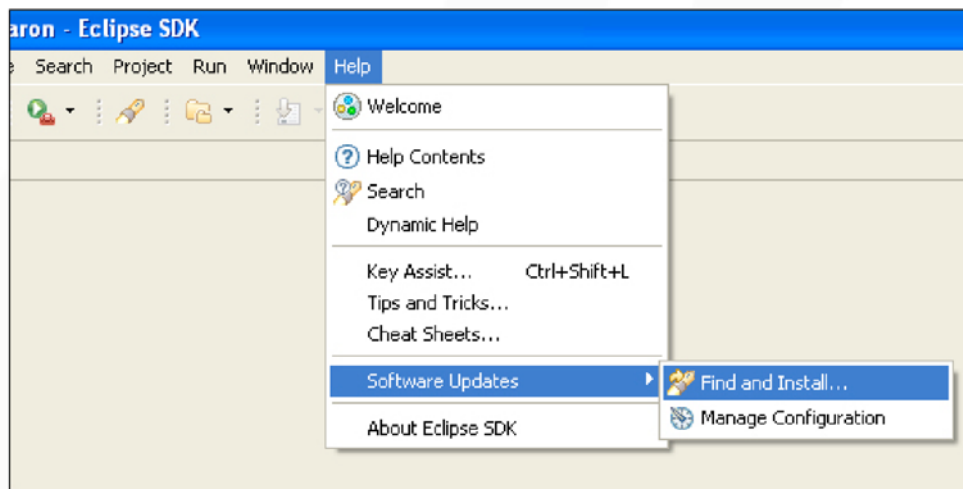


Figure 65: Find and Install Eclipse Plugins

The first step to installing most Eclipse plugins is to first access the *Find and Install* option under *Help – Software Updates*. This menu option allows you to create a new update site where Eclipse will look for installation files for plugins you already have installed or are interested in installing.

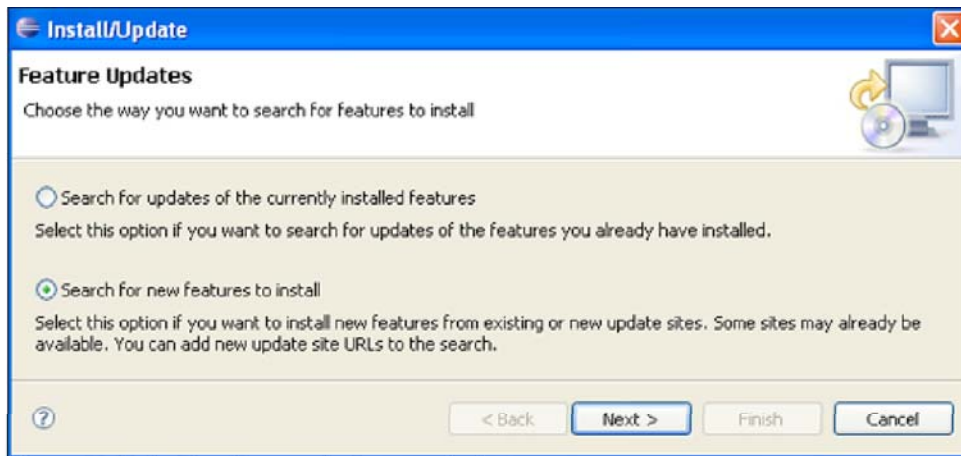


Figure 66: Selecting the new features radio button

You can elect to search for updates to currently installed features or search for new features to install. Since we're installing Subclipse for the first time, select the *Search for new features to install* radio button and press *Next*.

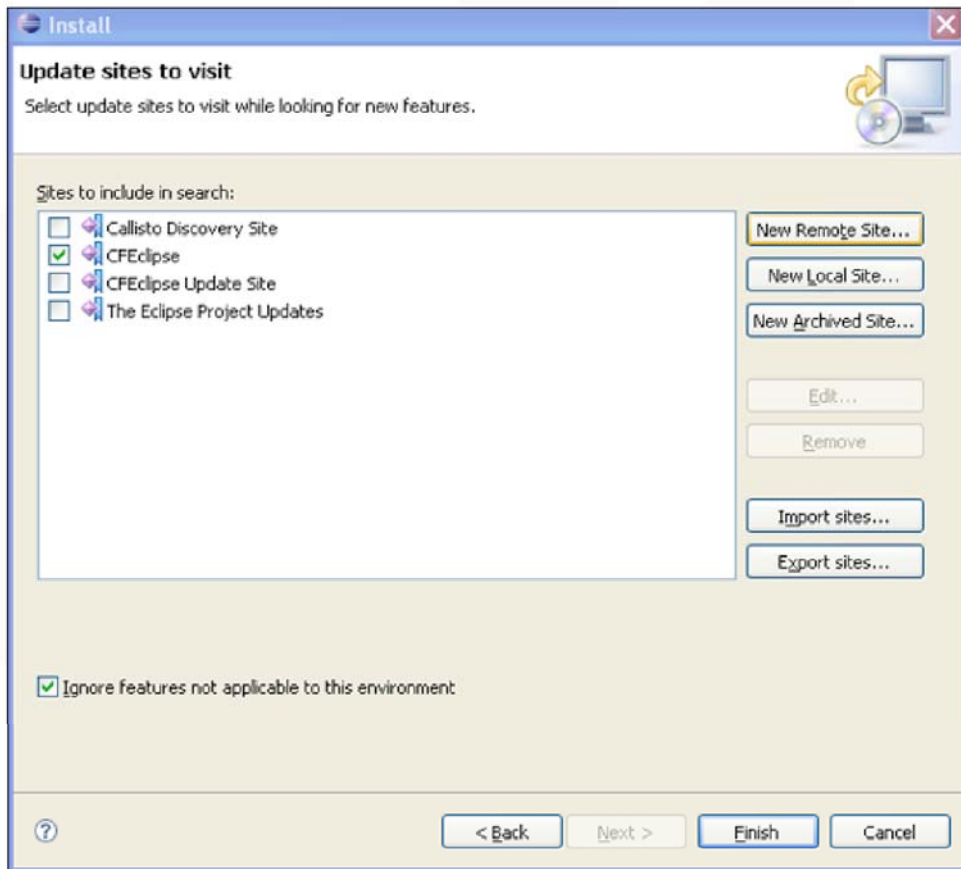


Figure 67: Current remote update sites

Eclipse shows you a list of all the remote update sites you have configured. Press the *New Remote Site* button to add a new site for Subclipse.

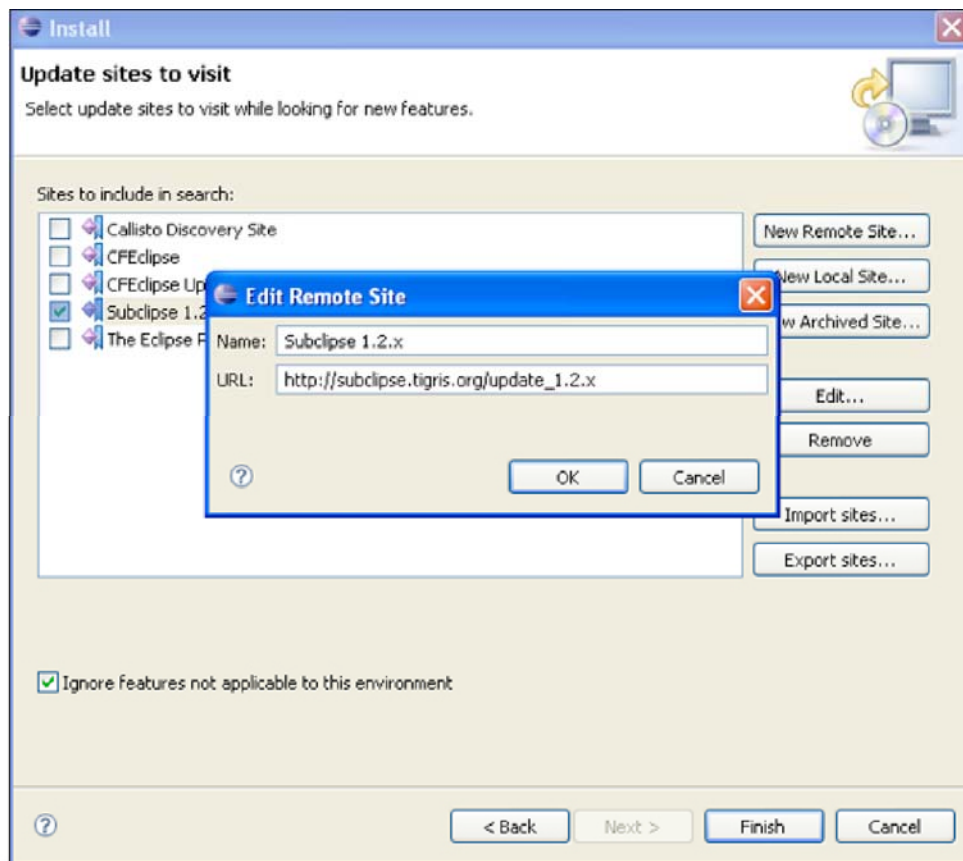


Figure 68: Adding the Subclipse update site

In the modal window that appears, enter a name for the update site (*Subclipse 1.2.x*) and the update URL (*http://subclipse.tigris.org/update\_1.2.x*). Press *Ok*.



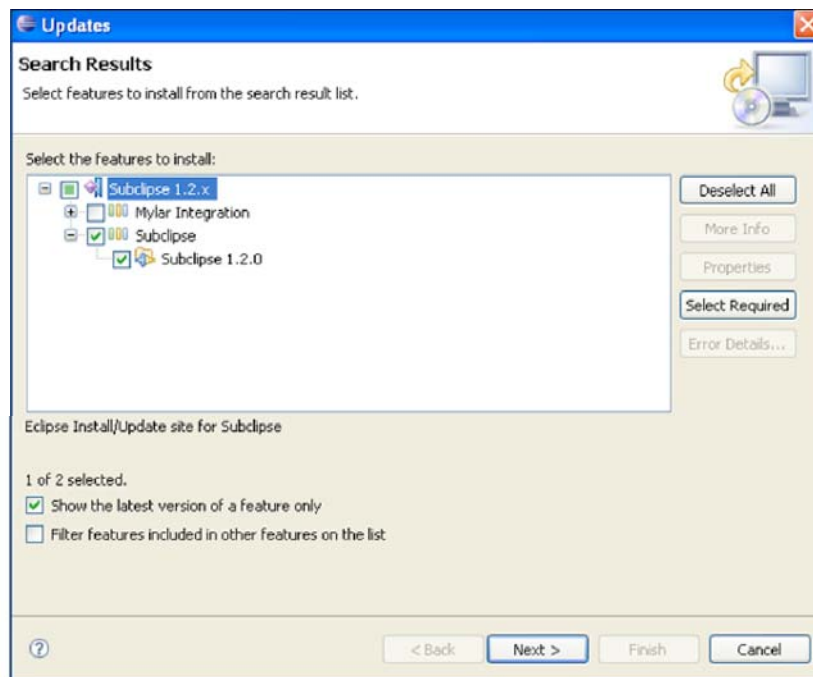


Figure 69: Selecting the version of Subclipse to install

A new Subclipse site should be listed in the Updates window and it should be checked. Expand the entry until you see the versions of Subclipse listed. Only the latest version of Subclipse will be listed if the check box at the bottom of the window for “show the latest version only” is selected. Select the latest version and press *Next*.

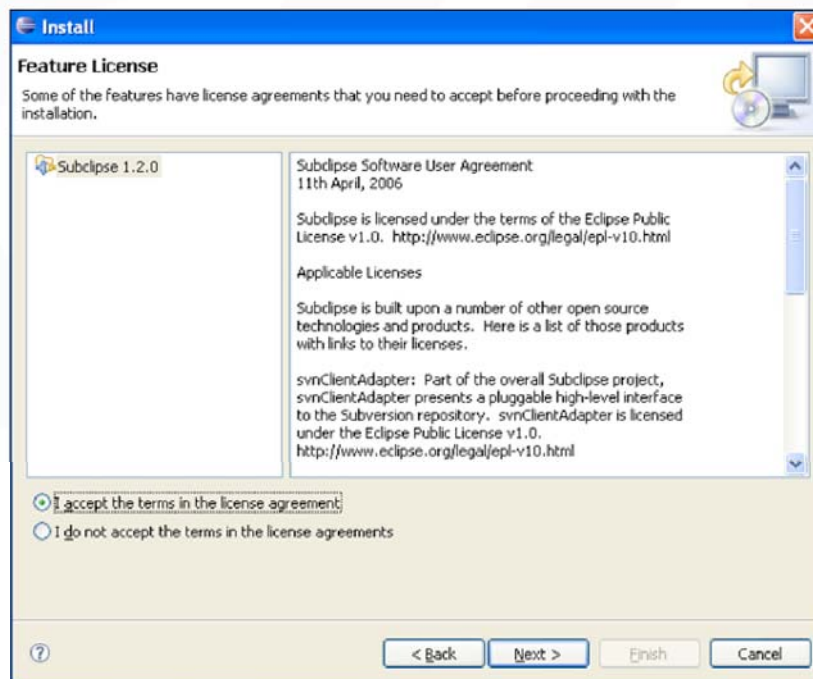


Figure 70: Subclipse License Agreement

Accept the terms of the Subclipse license agreement and press *Next*.

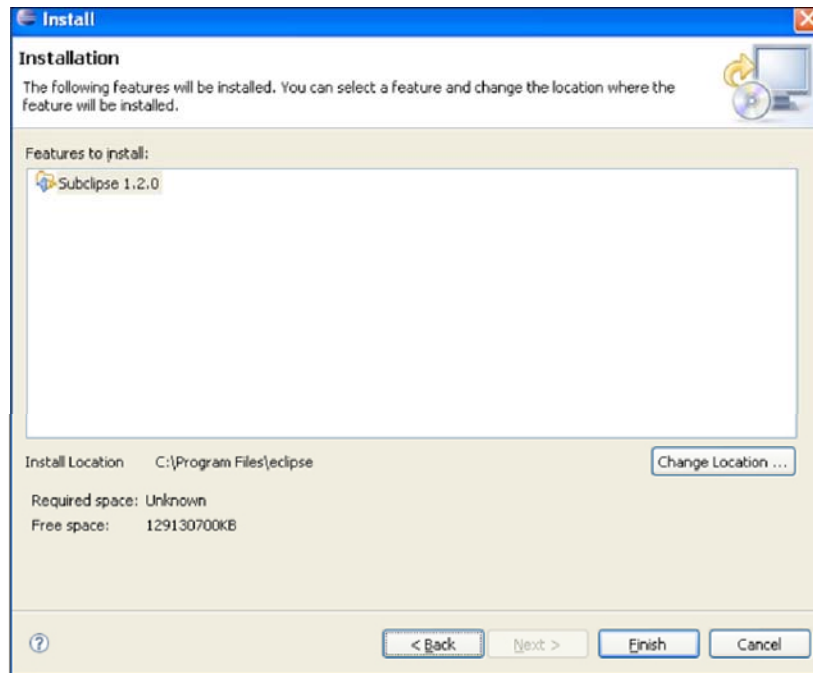


Figure 71: Subclipse installation location

Subclipse needs to be installed in the folder where Eclipse is installed. By default this is *C:\Program Files\eclipse*. Press *Finish* and the Subclipse installation files will be downloaded.

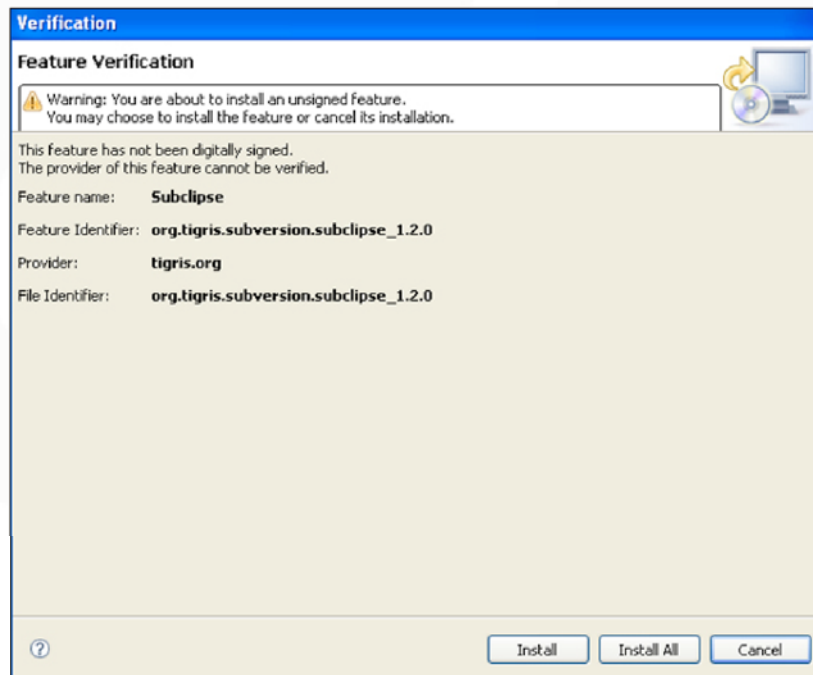


Figure 72: Subclipse installation verification screen

Press *Install* to begin the installation of Subclipse.

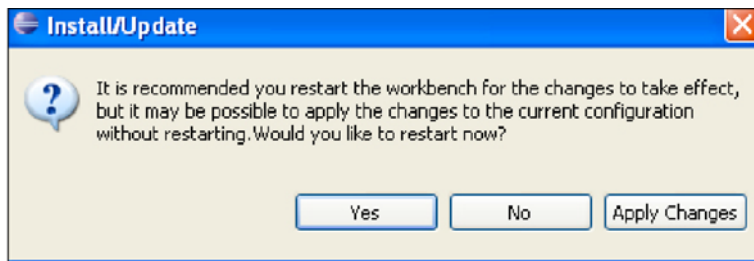


Figure 73: Restarting the workbench

In order to finish the installation of Subclipse you should restart the Eclipse workbench. Press Yes to do so.

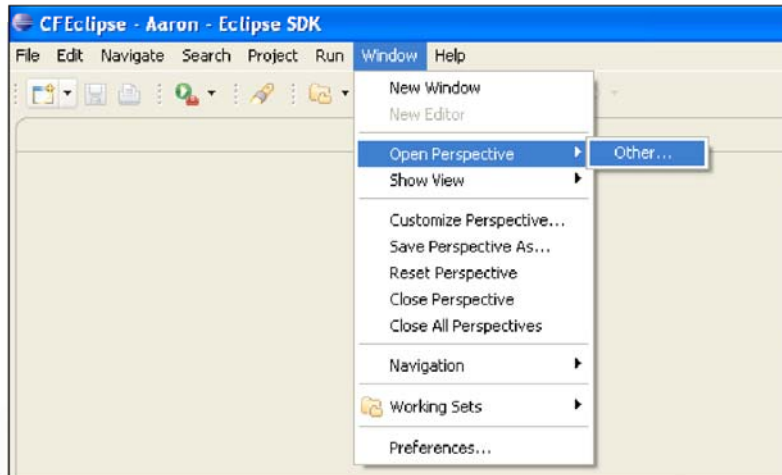


Figure 74: Verify the installation of Subclipse

To verify Subclipse was installed successfully open the perspectives window using Figure 74 as a guide.

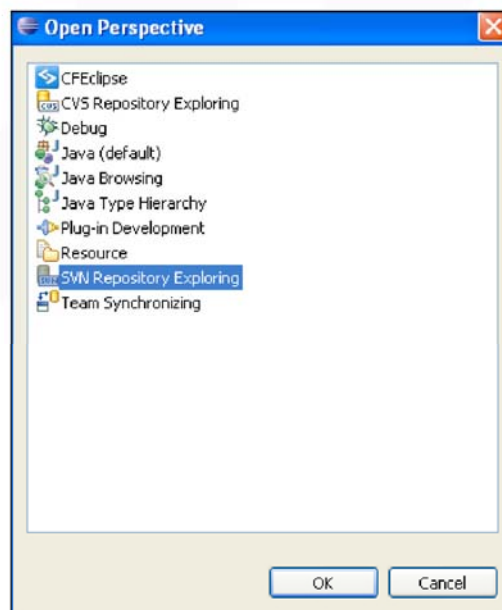


Figure 75: SVN Repository Exploring

Subclipse is installed and available if the *SVN Repository Exploring* perspective is listed.

### *Creating an Eclipse Project Connected to a Subversion Repository*

With Subclipse installed we can now create a repository location inside of Eclipse that points to the Subversion server we've set up in previous parts. Select the *SVN Repository Exploring* perspective and press *Ok* to open it.

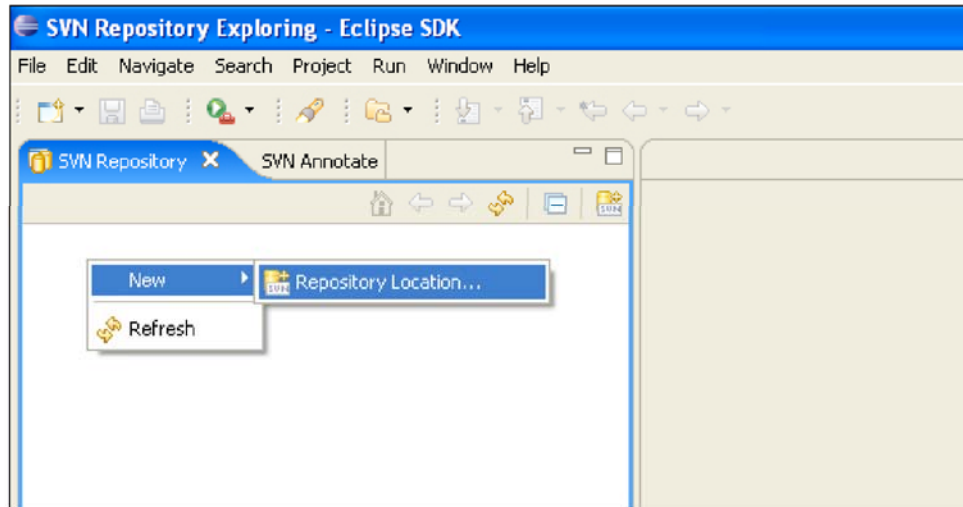


Figure 76: Creating a repository location

If the *SVN Repository* tab (which is a “view”) does not display, select *Window – Show View – SVN Repository*. Right-click the empty space in the *SVN Repository* tab and select *New - Repository Location*.

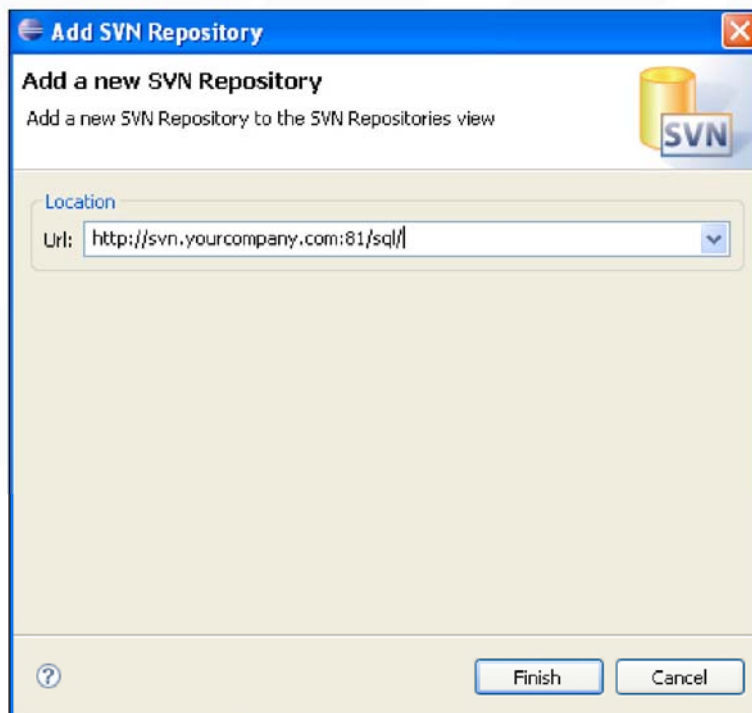


Figure 77: Specifying the repository URL

To work with repositories, we need to tell Subclipse where they are located. Type the URL to the sample repository we've been working with throughout this text (<http://svn.yourcompany.com:81/sql/>) and press *Finish*.

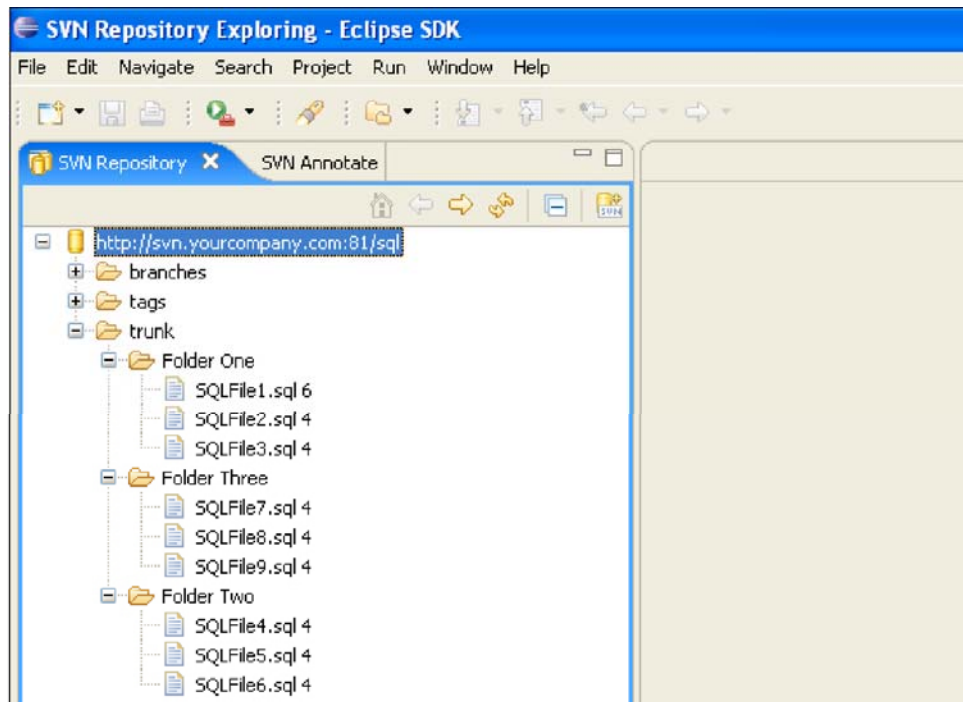


Figure 78: SQL repository contents

Subclipse introspects the repository URL you entered and shows the repository contents. Expand the repository to the trunk folder to ensure the contents of the repository are listed correctly.

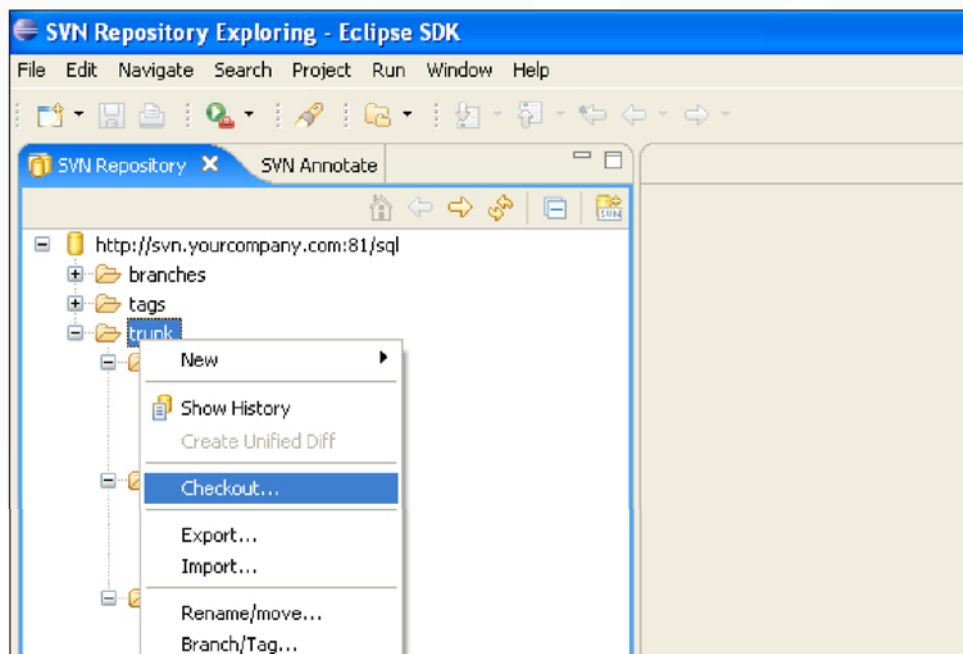


Figure 79: Checking out the repository to a local working copy

In order to begin working with the repository we must create a local working copy. In the next few steps we'll direct Subclipse to create a new Eclipse project that points to the *trunk* folder of our sample SQL repository. Start this process by right-clicking the *trunk* folder and pressing *Checkout*.



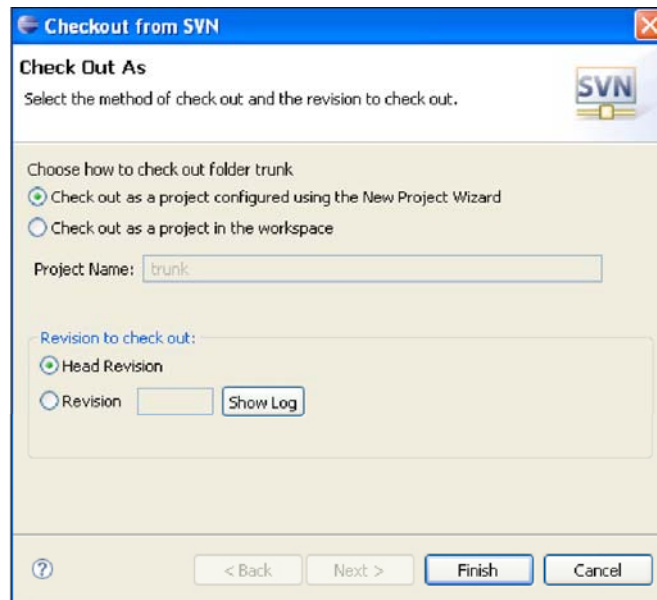


Figure 80: Checking out the repository as a new Eclipse project

Next, select the first option for checking out the *trunk* folder as a new project using the project wizard. Press *Finish* to continue.

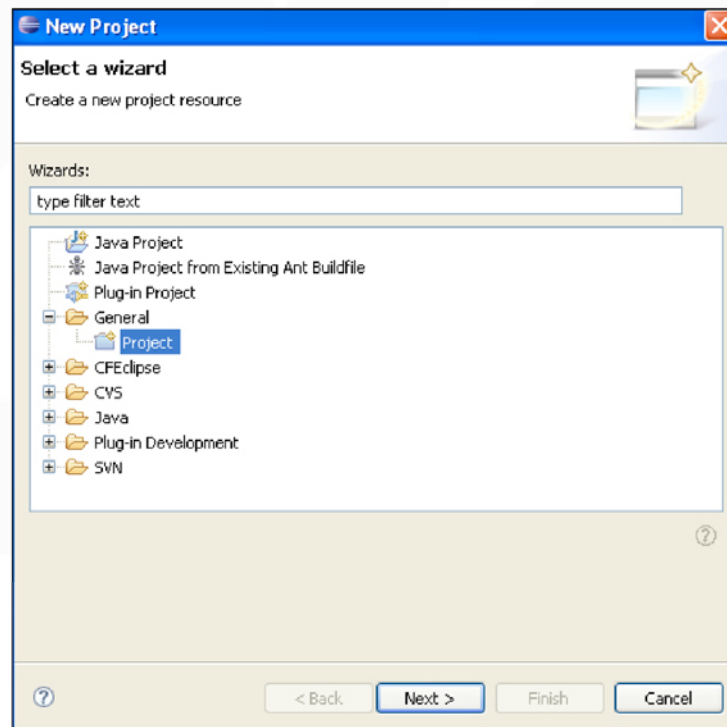


Figure 81: Selecting the project type

This screen allows you to select the type of project you are creating. Since our project relates to SQL files, just select the *General Project* type. Press *Next*.

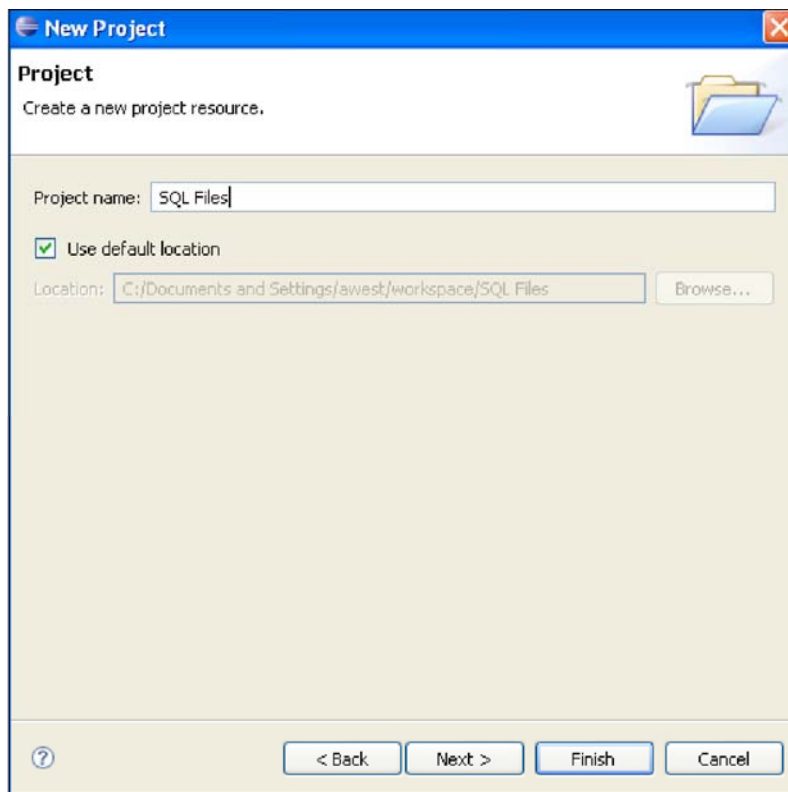


Figure 82: Naming the new project

We're now prompted to enter a name for our project. Keeping with the same names we've been using thus far, give the project the name *SQL Files* and select the location. The default location is fine unless you have a need to store your files elsewhere. Press *Next* to advance to the Project References screen. Our project doesn't need to reference any existing Eclipse projects so leave all boxes unchecked and press *Finish*.

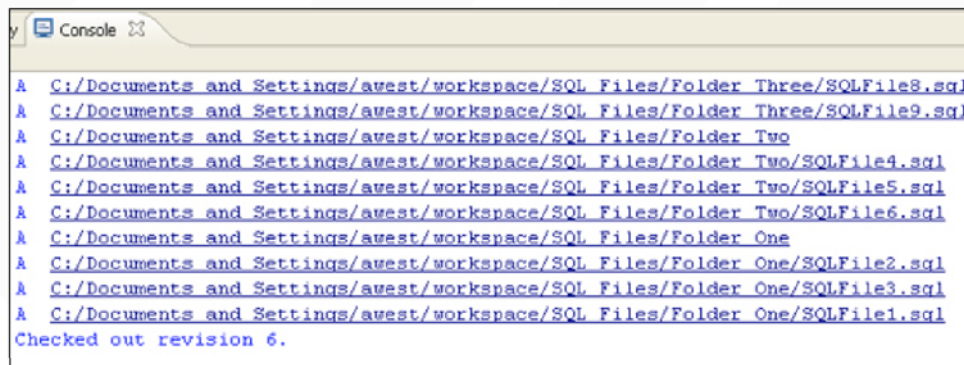


Figure 83: Console tab displays checkout log

After pressing *finish* Eclipse will create the new project in the *Navigator* tab and Subclipse will access the remote repository and check out the *trunk* to the root of the new project folder. If you have the *Console* tab open, you should see log messages from the checkout action performed by Subclipse.



Figure 84: New project listed in the Navigator tab

Switch to the *Navigator* tab and you should see the new *SQL Files* project listed. Expand the folders and you'll see all the files Subclipse checked out as well the version number for files. Also shown are special Subclipse icons next to folders and files indicating their status in relation to the repository.

## Committing Changes to the Repository

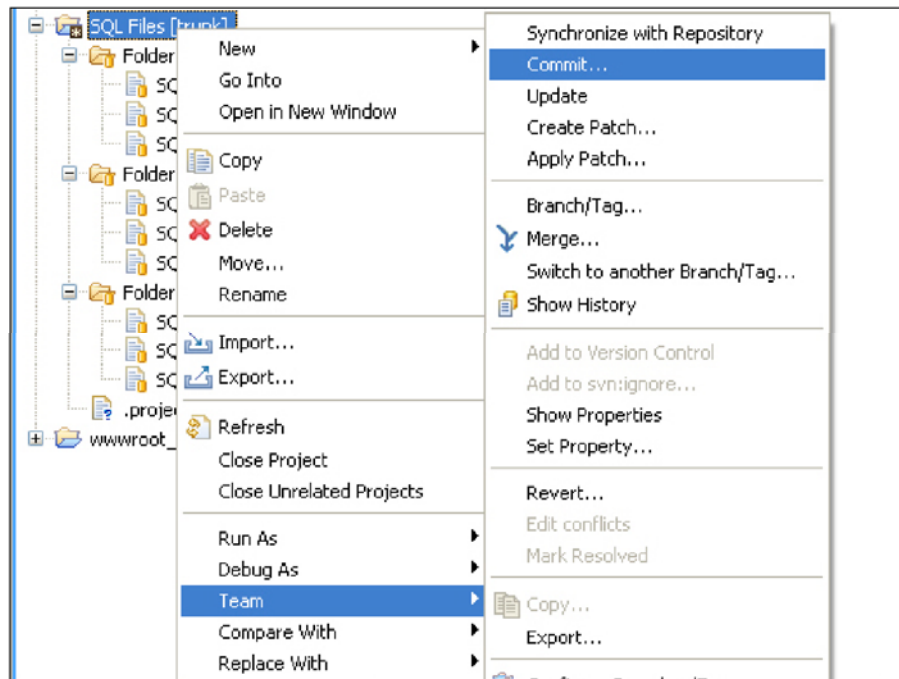


Figure 85: Subclipse Team window

In order to work with the Subversion repository you need to access the Subclipse commands. Much like TortoiseSVN these commands are located in a right-click window – well, most of them at least. Right-click the project folder and select the *Team* option. A new menu will appear showing all the things you can do. You can commit changes, update your local working copy with changes from the repository, show the history of the repository, create branches and tags, and revert to previous versions just to name a few options. To see how commits work, open the *SQLFile1.sql* file and make a change.

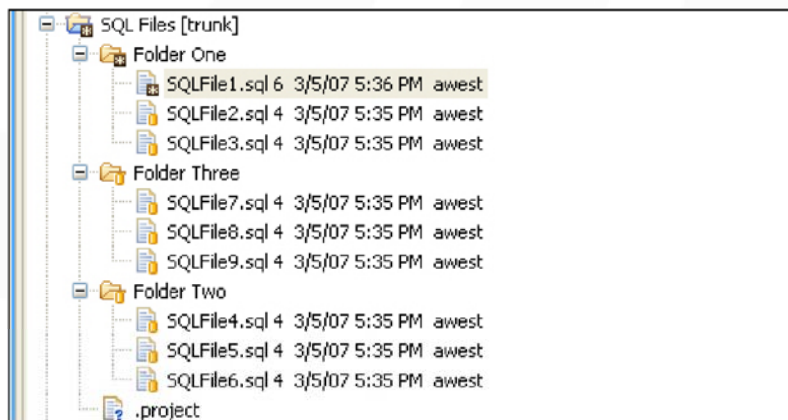


Figure 86: SQLFile1.sql has been changed locally

Once you've saved your change, an asterisk icon will display to the left of *SQLFile1.sql* indicating it has been changed locally and differs from the same file in the repository. To commit your change you have two options. You can right-click the file you changed and select *Team – Commit*. This will allow you to commit only that one file. To see ALL files that have changed locally, right-click the project folder and select *Team – Commit* (Figure 85).

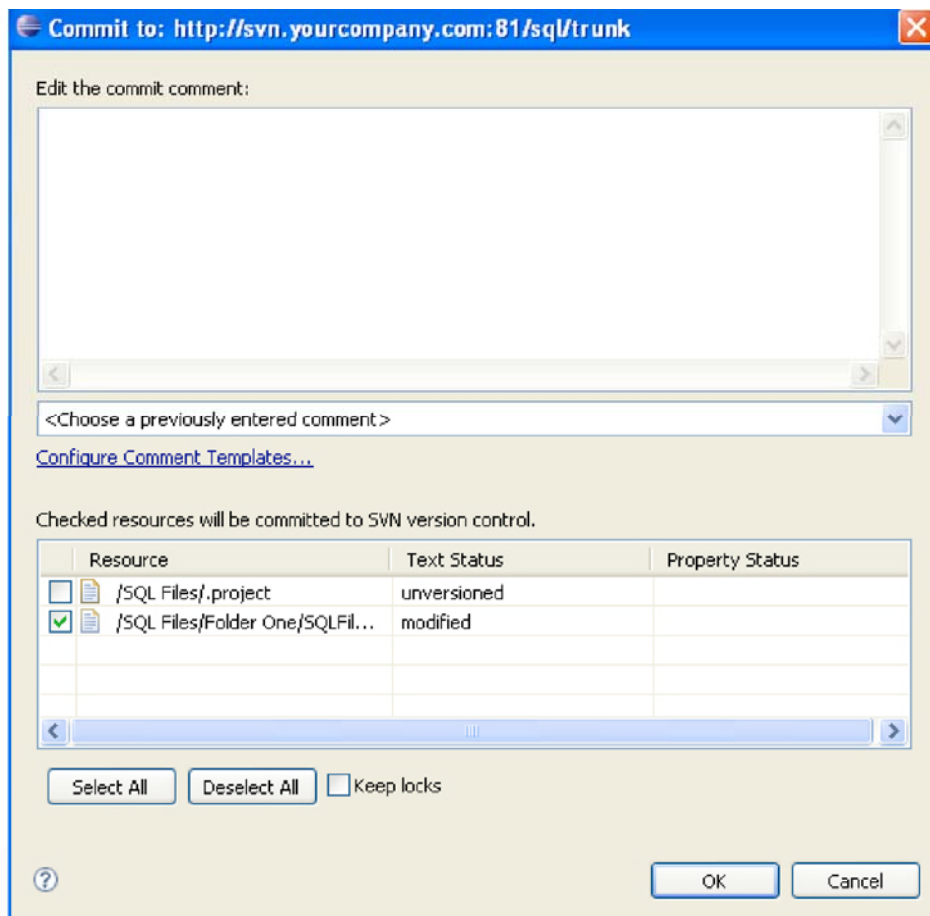


Figure 87: Selecting files to commit

The bottom pane will show you any file that can be committed. You can ignore the .project file. Make sure only *SQLFile1.sql* is checked, enter a comment, and press *Ok*.

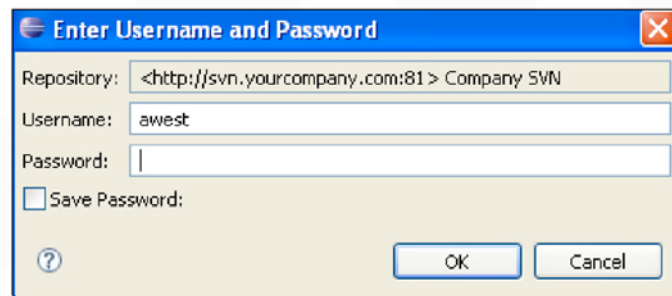


Figure 88: Authenticating with the repository

Subclipse will prompt you for your username and password (if you followed the steps in Part 4). Enter your username and password and press *Ok*. You can elect to have Subclipse save your password if you don't want to type it over and over again. You will definitely want to do this at some point, as it gets very annoying typing your password for each and every “authentication-required” action taken on the repository.



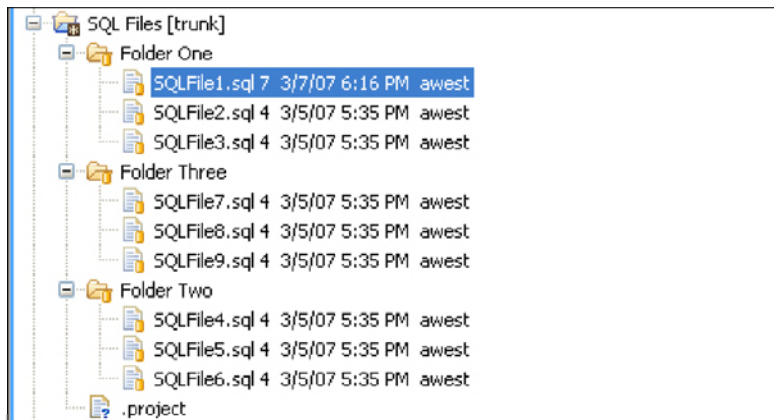


Figure 89: Commit results

Once the commit action has finished the version number listed to the right of *SQLFile1.sql* should increment to 7 (based on the actions we've taken thus far in the various parts of this text). The *Console* tab, if you still have it open, will list the Subversion command used to perform the commit and the results (a new version number).

### Getting Changes from the Repository

Repository updates work pretty much the same as commits, you just select the *Update* option of the *Team* menu after right-clicking the project directory. To view the latest actions taken on the repository you can display the repository history. Right-click the project directory in the *Navigator* pane and select *Team – Show History*.

Tasks	Problems	Bookmarks	Browser View	Ftp Log View	Outline	Methods View	History
SQL Files							
R...	Tags	Date	Author	Comment			
*7		3/7/07 6:16 PM	awest	Changed the sql file.			
6		3/5/07 5:36 PM	awest	I've made an additional change.			
5		3/5/07 5:35 PM	awest	I made a change!			
4		3/5/07 5:35 PM	awest	Initial import of sql code.			
1		3/5/07 5:32 PM	awest	Created folder remotely			
Act...	Affected paths		Description				
M	/trunk/Folder One/SQLFile1.sql						

Figure 90: Viewing the repository history

The *History* tab will list the history of actions taken on the repository. This includes actions you've taken using TortoiseSVN and Subclipse as well as any actions taken by others.

## Synchronizing a Working Copy with the Repository

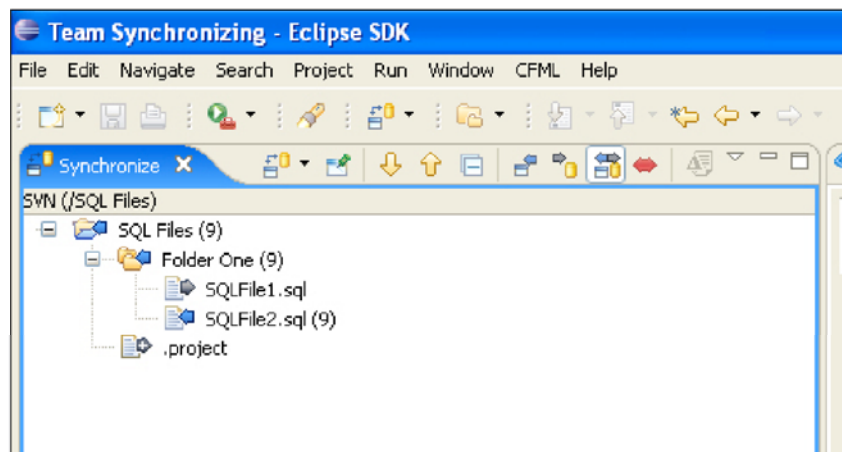


Figure 91: Team Synchronizing

One final feature of Subclipse I want to discuss is the *Synchronize with Repository* option. Select this option by right-clicking the project folder and selecting *Team – Synchronize with Repository*. Subclipse will let you know the feature is part of the Team Synchronizing perspective in Eclipse. Press *Yes* to continue. The *Synchronize with Repository* option allows you to view changes you need to download (through an *Update* request) and changes you need to upload (through a *Commit*). An example is shown in Figure 91. As you can see, I have made a change to *SQLFile1.sql* while someone else has changed *SQLFile2.sql*. To update my local copy - receiving the new *SQLFile2.sql* file - and commit my own changes I have a few options. First, I could right-click individual files I need to download and select *Update* in order to retrieve them. Or, to download all updates, I could right-click in the whitespace and select *Update*. Commits work very much the same. I could right-click individual files I need to commit and select *Commit*, or I could right-click the project folder and select *Commit* to commit all my local changes at once. In either case, Subclipse will prompt me to enter comments before sending my changes to the repository.

### Summary

In this part, I walked you through installing Subclipse and creating a working local copy of the sample repository files. I also discussed the most commonly used features of Subclipse: updating your working copy, committing your own changes, showing the repository history, and synchronizing between your working copy and the remote repository. I'm a pretty big fan of Subclipse and the Eclipse environment in general. I use the Eclipse platform on a daily basis to manage workplace code and personal code. However, TortoiseSVN is also a nice Subversion client – especially when used on the server that hosts the Subversion server and repositories. Add these tools to an “Apache-based” Subversion environment and you have, in my opinion, a very nice development set up on the server-side and client-side.

I hope this text has provided you with a good overview of how to set up and configure your development environment with Apache, Subversion, TortoiseSVN, and Subclipse. As I said in the introduction and disclaimer, the implementation I describe is not the only way to configure a source control environment. I encourage you to consult other resources on how to set up and further refine your own environment. To that end, I have included various resources that I feel are useful on this topic. Some of these I used as a reference in deciding how to configure the development environment where I work. Others are useful, in general, for anyone wanting to configure a localhost Web development set up. Finally, there are several resources listed that expand on many of the ideas I presented in this text. If you're interested in learning more about source control management I highly recommend checking them out. Good luck, and have fun!

## Additional Resources

### *Books*

Collins-Sussman, B., Fitzpatrick, B., & Pilato, M. (2004) Version Control With Subversion  
O'Reilly Media  
<http://svnbook.red-bean.com/>

Collins, S. (2006) The ACME Guide (3<sup>rd</sup> ed.)  
Stephen Collins  
<http://acidlabs.org/extras/acme>

Mason, M. (2006) Pragmatic Version Control Using Subversion (2<sup>nd</sup> ed.)  
Pragmatic Bookshelf  
<http://www.pragmaticprogrammer.com/titles/svn/>

### *Links*

Subversion FAQ  
<http://subversion.tigris.org/faq.html>

Subversion Cheat Sheet  
<http://www.abbeyworkshop.com/howto/misc/svn01/>

Charlie Arehart's Subversion Resource List  
[http://carehart.org/blog/client/index.cfm/2006/7/12/subversion\\_resources](http://carehart.org/blog/client/index.cfm/2006/7/12/subversion_resources)

Dave Shuck on setting up Subversion for his development team.  
<http://www.daveshuck.com/blog/index.cfm/2006/10/2/New-Subversion-development-environment>

Rob Gonda explains differences between SVN Serve and Apache.  
<http://www.robgonde.com/blog/index.cfm/2006/8/2/SVN Serve-Vs-Apache>

Brian Kotek on using Subversion (svnserve) locally.  
<http://www.briankotek.com/blog/index.cfm/2007/1/3/Overview-of-Using-Subversion-Locally>

Net Batchelder discusses Subversion branching strategies.  
<http://www.nedbatchelder.com/text/quicksvnbranch.html>

Brandon Harper discusses best practices for Subversion branching.  
<http://devnullled.com/content/2006/10/guide-and-best-practices-for-subversion-branching/>

Rick Osborne's CFDIFF  
<http://rickosborne.org/blog/?p=86>

### *Software*

Apache HTTP Server  
<http://httpd.apache.org>

Subversion Server Packages  
[http://subversion.tigris.org/project\\_packages.html](http://subversion.tigris.org/project_packages.html)

TortoiseSVN  
<http://tortoisesvn.tigris.org/>

Eclipse  
<http://www.eclipse.org>

Subclipse  
<http://subclipse.tigris.org>

CFEclipse (ColdFusion plug-in built on the Eclipse Platform)  
<http://www.cfeclipse.org>

