Introduction to Subversion

Wesley Smith

Media Arts and Technology, University of California, Santa Barbara
Santa Barbara, CA 93106, USA
whsmith@mat.ucsb.edu

1 Introduction

Subversion is a version control system based on the *copy-modify-merge* paradigm. The basic cycle involves each developer downloading a copy of the repository to their machine, called a *working copy*, making some modifications, and finally merging those changes back with the central repository. The best resource for learning about Subversion is the free book *Version Control with Subversion* available at http://svnbook.redbean.com/. There you will find a conceptual overview of the Subversion system, which is essential to using it productively. In addition, the book contains a thorough technical examination of setting up a Subversion repository, integrating it into your coding practices, and managing the repository over the lifetime of the projects it holds.

This manual is mean to be a brief primer on Subversion to get people who are not only unfamiliar with Subversion but with source code management systems in general up to speed and working productively as soon as possible. The following sections are based on gathered experience and knowledge from using Subversion for the past few years in a variety of contexts from small personal projects where I am the only author to large commercial projects with thousands of files and many authors simultaneously submitting modifications nearly every day.

1.1 Getting Started

Setting up a Subversion repository is a simple as executing a single shell command. Configuring the Subversion server, however, takes a bit more work, but more than likely the server hosting your repository has already been setup as a Subversion server so it's likely that you won't need to worry about this. On the MAT network, maize has already been configured to run the Subversion server.

1.2 Setting Up the Repository

If you are setting up a repository for personal use, you should create it in your home directory on maize. If you are setting up a group project, you should set it up in /mat/groups so that everyone can access it. You probably won't have write permissions in the groups folder, so ask the sysadmin (i.e. Larry) to hook you up.

Subversion has two basic basic commands (there are actually 9, but two are much more frequently used). They are *svn* and *svnadmin*. *svn* is used to manage files located

in the repository while *svnadmin* deals with the repository directly. For setting up the repository, we are going to use *svnadmin*. First, pick a directory where you want to put your repository. Something like /repos is a good choice. If it doesn't already exist, create it.

On the server (maize.mat.ucsb.edu):

```
$ cd
$ mkdir repos
$ cd repos
```

Now that we have a folder where we want to put our repository, we can create it using *svnadmin*:

```
$ svnadmin create myrepo
```

This command will create a new repository called 'myrepo' (substitute your own name). You're now done! If you list the files in the new repository, you should see something that looks like:

README.txt
conf
dav
db
format
hooks
locks

1.3 Setting Up the Working Machine

The Subversion server can be configured to run in a number of different modes allowing access through HTTP as well as SSH. In my experience, HTTP is slower than SSH and thus not preferable. On maize, The Subversion server is configured to run under SSH. Access to a Subversion repository through SSH is through a specially defined URL that subversion understands, taking the format svn+ssh://username@server/path/to/repo. For example, if your name is Hackme, your URL for the repository we just created would be svn+ssh://hackme@maize.mat.ucsb.edu/home/hackme/repo/myrepo. This is the URL you use from your working machine to access the repository. However, since we're using SSH, we'll need a password each time we try to access the repository. This can get very annoying very quickly. Fortunately, there's a way to automatically login through SSH using what's called SSH key authentication. If you have already setup SSH key authentication, you can skip this next section and move on to section 1.4.

SSH Key Authentication SSH key authentication allows one computer to login to another through a generated key instead of the usual username/password system. Keys can also require a username and password and many sites recommend generating keys this way. For our purposes we are going to generate a key that doesn't require a password as we don't want to have to type it in all of the time.

To setup SSH key authentication, we first generate a key on the working computer, creating both a public and private key. The private key stays on the working computer. The public key goes on the server you want to login to (i.e. maize.mat.ucsb.edu). Once properly setup, you should be able to login to this machine (and use *scp*, *svn*, etc) without a password. There are a number of steps to this process some of which you may not need to do depending on what is already present on your machines. The instructions below are for *NIX systems. Windows people should consider throwing their machines out the window or at least installing a real OS with a real command line like Ubuntu.

The first step is to generate a public/private key pair. From your home directory on your working mahcine:

```
$ ssh-keygen -trsa
```

You will be asked for a filename. Just use the default (i.e. ~/.ssh/id_rsa and hit 'enter').

Next you will be asked for a password. Don't type one in and hit 'enter' again twice. Finally, you will see confirmation that the key generation command has succeeded and created two files in your .ssh folder: id_rsa and id_rsa.pub. This last one is the public key we will upload to the server.

To get the key to the server, copy the public key to your home directory:

```
$ cd .ssh
$ scp id_rsa.pub username@servername:~/
```

i.e. scp id_rsa.pub hackme@maize.mat.ucsb.edu:~/

Now, log in to the server to configure the key properly:

```
$ ssh username@server
```

On the server, we need to construct a .ssh directory if it doesn't exist and place the public key where SSH will be able to find it during the login process. If you don't have a .shh folder, do the following:

```
$ mkdir .ssh
$ chmod 755 .ssh
```

If you don't have a file called authorized_keys in your .ssh file, do the following

```
$ touch .ssh/authorized_keys
```

To configure the public key and ensure correct permissions, do the following:

```
$ chmod 600 .ssh/authorized_keys
$ cat id_rsa.pub >>.ssh/authorized_keys
```

At this point, you should have a properly configured setup with the recently uploaded public key now copied into the end of your authorized_keys file. To test it out, open a shell on your working machine and try to login:

```
$ ssh username@server
```

If it asks you for a password, something is not working properly. More than likely this is due to there not being a line-break between successive public keys on the server's authorized_keys file. After each key, there is an '==' and some text marking its origin. This should end a line of text. If there is more after it that looks like a bunch of encrypted numbers, add a line-break in the appropriate spot and try again.

1.4 The First Commit

At this point, we are ready to put the first files in the repository. If the repository already has files in it, skip this section and go to **??**. First, a little conceptual background on Subversion. Subversion has been designed to make branching a project a core part of the development process of a project. A branch is a parallel version of a project with potentially significant differences. Depending on the project and the point in the development cycle, a branch could be created for a number of reasons. The most common are:

- To spawn a subproject from a larger main project that will diverge over time
- To keep the main branch (aka the *trunk*) in working order while someone rewrites a significant section of code that will introduce incompatibilities

This last reason is the most common and allows a team of developers to keep pushing a project forward despite a section of code being introduced that would otherwise break the build. In the end, this rewrite will be merged back into the *trunk*. Subversion keeps a complete record of the entire repository structure for each revision which increments on each commit. As a result, Subversion provides a number of tools for automating the merge and update process.

The reason for bringing up the branching concept at this point is that it affects repository layout. The canonical subversion layout includes root folders called 'branches', 'trunk', and 'tags'. If more than one project is placed in a repository, then each project should have'branches', 'trunk', and 'tags' folders. To initialize the repository with this layout, create a new folder on your working machine where you want to host your working copy. In this folder, create the folders "branches', 'trunk', and 'tags'. It should look like this:

/localrepofolder
  /branches
  /tags
  /trunk

Now we need to commit this to our repository on the server. The easiest thing to do is use the *svn import* command. In a terminal, go to the location of the folders you want to add to the repository. To add them, do the following:

```
$ svn import foldertoadd svn+ssh://username@server/path/to/repo
-m "initial commit"
```

The -m switch is for attaching a message to the import action so you know what you were thinking at the time. You will get an error without it. Now that we've imported the file structure to the repository, we need to make a local working copy because the import command does not convert the imported files to a working copy. To get a copy of the repository, go to a folder that you want to work from and type:

```
$ svn co svn+ssh://username@server/path/to/repo
```

You should see the list of files you just imported listed as they are downloaded along with the text "Check out revision 1". Now you have a working copy. This means that subversion knows the location of the repository on the server and you don't have to type it in explicitly anymore. From anywhere within your working copy, you can commit and update and Subversion will know where to put the files.