

Subversion on CentOS

Subversion isn't just for coders and programmers...I know because I'm not one. I've begun to use it lately for many things, such as, backing up Nagios configurations, documents, and pretty much anything text based. I don't know why I didn't start using it sooner, but none the less, here I am. This document quickly explains how to install, configure, and use subversion locally, as well as across Apache on a network. For complete and complex configurations and installations seek the documentation provided. There is plenty of well written documentation on the subject, that far exceeds my knowlegde of the tool. This just helps get you started quickly, for those like me that just like to jump in head first into new things.

Contents

1. System
2. References
3. Installation
4. Configurations
5. Using Subversion
6. Access control lists
7. Afterthought
8. Further reading

1. System

CentOS 4.x/RHEL 4

CentOS 5.1/RHEL 5

2. References

Subversion: <http://subversion.tigris.org/>

Version Control with Subversion: <http://svnbook.red-bean.com/>

3. Installation

```
[root@lucifer ~]# yum install mod_dav_svn subversion
```

The first thing to do is to install the packages I mentioned above. If you don't have Apache installed already, it'll go ahead and drag that down as well.

When you install from yum, there's a longer list than the two packages above that will automatically resolve themselves. Some other things will be installed automatically. Depending on your packages, your mileage may vary.

4. Configurations

4.1. Apache

Before you delve into the deep end, you need to ensure Apache is set up first. I'm assuming this is a virgin installation, so if you already have Apache things going...be careful what you change. I'm also going to explain setting this up with basic password protection. You can easily let this out, however, if you want to allow access to the repos from everyone.

First thing is make sure you open up `/etc/httpd/conf/httpd.conf` and at least change the `ServerName` directive. If you need more help or more complex configurations, then consult the Apache docs please.

```
[root@lucifer ~] vim /etc/httpd/conf/httpd.conf -- Edit what you
need and save the file
[root@lucifer ~] service httpd start
[root@lucifer ~] chkconfig httpd on
```

Browse to your machine on the network and see if you get your test page, which you should: <http://yourmachine>. Working? Great, let's move along to more fun things.

4.2. Subversion's Apache configs

The next step is to setup some settings within Apache so Subversion and Apache play nice together. Get yourself to the example configuration file Subversion installed for you.

```
[root@lucifer ~] cd /etc/httpd/conf.d/
[root@lucifer ~] vim subversion.conf

# Make sure you uncomment the following if they are commented out
LoadModule dav_svn_module      modules/mod_dav_svn.so
LoadModule authz_svn_module    modules/mod_authz_svn.so

# Add the following to allow a basic authentication and point
# Apache to where the actual
# repository resides.
<Location /repos>
    DAV svn
    SVNPath /var/www/svn/repos
    AuthType Basic
    AuthName "Subversion repos"
    AuthUserFile /etc/svn-auth-conf
    Require valid-user
</Location>
```

The location is what Apache will pass in the URL bar. For instance: <http://yourmachine/repos> points to the `SVNPath` that you have specified. My examples are just that, so feel free to put things where you want. Make sure you save the file when you are finished editing.

Next we have to actually create the password file that you specified in the previous step. Initially you'll use the `-cm` arguments. This creates the file and also encrypts the password with MD5. If you need to add users make sure you simply use the `-m` flag, and not the `-c` after the initial creation.


```
[root@lucifer ~] htpasswd -cm /etc/svn-auth-conf yourusername
New password:
Re-type new password:
Adding password for user yourusername
[root@lucifer ~] htpasswd -m /etc/svn-auth-conf anotherusername
New password:
Re-type new password:
Adding password for user anotherusername
```

4.3. Configure your repository

The next thing you need to do is to create the actual repository from which you will check in and out your files. This is simple to do with some of the included svn tools.

```
[root@lucifer ~] cd /var/www/ -- Or wherever you placed your path
above
[root@lucifer ~] mkdir svn
[root@lucifer ~] cd svn
[root@lucifer ~] svnadmin create repos
[root@lucifer ~] chown -R apache.apache repos
[root@lucifer ~] service httpd restart
```

Go test out whether or not you can access your repository from a web browser:

 <http://yourmachine/repos>. You should get a popup box asking for a username and password. If so, type in your credentials and you should be displayed with a *Revision 0:/* page. If so, that's it for setting up a repo. If you want multiple repos, check out the docs from the links provides above. This sets up one repository and shows you how to start using them. Speaking of, let's move on to just that.

5. Using Subversion

5.1. Layout Your Repo

If all went well above, you're now ready to start using the repository that you created. Subversions `svn` tool is the command line client that you will use to talk to the database. To see the use of the tool:

```
[root@lucifer ~] svn --help
```

The most common arguments you will most likely be using are: `svn import`, `svn commit (ci)`, and `svn checkout (co)`. With these you will initially import files into your repository with `import`, you'll check them out to work on them with `checkout`, and you'll commit the changes back into the database with `commit`. It's pretty simple once you see them in use a few times.

Before I continue I'd like to explain about directory structure layouts. Almost all of the

documentation talks about creating a certain layout for your directories. They specifically mention about making sure you have a branches, tags, and trunk underneath the root directory structure, where trunk holds all your files. For instance:

```
.
|-- project1
|   |-- branches
|   |-- tags
|   `-- trunk
`-- project2
    |-- branches
    |-- tags
    `-- trunk
```

The book explains why in a bit more detail, and I mainly don't bother using this type of layout...because I'm not doing coding and maintaining "projects" per se. I'm mainly using it to store configuration files, and text items that aren't as complex. Set things up for whatever works for you.

As an example, I'm going to just create some dummy directories and throw some files in them. This is from the actual SVN server. Play along.

```
[root@lucifer ~] cd /tmp
[root@lucifer ~] mkdir mytestproj
[root@lucifer ~] cd mytestproj
[root@lucifer ~] mkdir configurations options main
[root@lucifer ~] vim configurations/testconf1.cfg -- Add whatever
you want to these files.
[root@lucifer ~] vim options/testopts1.cfg
[root@lucifer ~] vim main/mainfile1.cfg
```

Keep in mind that you can layout anything anyway you'd like. Once you have the initial layout of what you want, let's go ahead and import this up to Subversion.

5.2. Importing

```
[root@lucifer ~] svn import /tmp/mytestproj/
file:///var/www/svn/repos/mytestproj -m "Initial repository layout
for mytestproj"
Adding          /tmp/mytestproj/main
Adding          /tmp/mytestproj/main/mainfile1.cfg
Adding          /tmp/mytestproj/configurations
Adding          /tmp/mytestproj/configurations/testconf1.cfg
Adding          /tmp/mytestproj/options
Adding          /tmp/mytestproj/options/testopts1.cfg
```

5.3. Checking Out

Now, just to check it out across the web browser: <http://yourmachine/repos>. You'll get whatever you have imported showing up to peruse. Once you upload your original layout from the local SVN server, you're now free to use it remotely on another machine. As long as you are connecting to the Subversion server with the user account(s) that you created earlier. Let's give it a shot.

```
[me@mylappy ~] cd /tmp
[me@mylappy ~] svn co http://yoursvnserver/repos/mytestproj
Authentication realm: <http://yoursvnserver:80>; Subversion
repos
Password for 'youruser':
A    mytestproj/main
A    mytestproj/main/mainfile1.cfg
A    mytestproj/configurations
A    mytestproj/configurations/testconf1.cfg
A    mytestproj/options
A    mytestproj/options/testopts1.cfg
Checked out revision 1.
```

5.4. Edit & Commit

As you can see, you've checked out revision 1 from the Subversion server. Now you can edit some things and commit the changes back to the Subversion server.

```
[me@mylappy ~] cd mytestproj
[me@mylappy ~] vim configurations/testconf1.cfg -- Add or delete
something and save.
[me@mylappy ~] svn commit -m "Added a line to testconf1.cfg."
Sending      configurations/testconf1.cfg
Transmitting file data .
Committed revision 2.
```

The nice thing about this then, is that you can delete all of the directories that you just checked out on your machine. The only reason you checked them out, was to edit them, and then send them back up the line. Web browse to your server to check out the different files.

5.5. Adding/Deleting Items

Now this is all fine and dandy, but how do you add more files to an already existing repo directory? Easy, with the add argument. Go ahead and checkout your latest and greatest, copy a file over to a directory, add, then commit the changes.

```
[me@mylappy ~] svn co http://yoursvnserver/repos/mytestproj
A    mytestproj/main
A    mytestproj/main/mainfile1.cfg
A    mytestproj/configurations
A    mytestproj/configurations/testconf1.cfg
A    mytestproj/options
A    mytestproj/options/testopts1.cfg
Checked out revision 2.

[me@mylappy ~] cd mytestproj
[me@mylappy ~] cp /etc/yum.repos.d/CentOS-Base.repo
configurations/
[me@mylappy ~] svn add configurations/CentOS-Base.repo
A    configurations/CentOS-Base.repo

[me@mylappy ~] svn commit -m "Added the CentOS Yum repo file."
Adding      configurations/CentOS-Base.repo
Transmitting file data .
Committed revision 3.
```

To delete items simply use delete instead of add. Commit your changes back up, and you're good to go. It's as simple as that. Go back over to your web browser again and you'll notice the revision number should say 3. You'll be able to click through the files to pick out your differences as well.

5.6. Reverting Back

Ok, this is all great but how do I revert back to an older revision...isn't this the point of Subversion? Yep, it's easy. If you're not sure as to what revision you're at...check out the log command. This is why you put a message in every commit. Short and to the point, but enough information to ring a bell that you perhaps forgot about.

```
[me@mylappy ~] svn log http://yoursvnserver/repos -- For the
entire repository
[me@mylappy ~] svn log http://yoursvnserver/repos/mytestproj --
For the specific project
```

You'll get a nice complete list of revision numbers along with the comments, like I mentioned above. This allows you to pick which revision you want to check back out now.

```
[me@mylappy ~] svn co -r 1 http://yoursvnserver/repos/mytestproj
```

This command will drag down revision number 1.

6. Access control lists

Usually, you don't want to give every user access to every repository. You can restrict repository access per user by using ACLs. ACLs can be enabled with the *AuthzSVNAccessFile* file option, which takes a file name as its parameter. For instance:

```
AuthzSVNAccessFile /etc/svn-acl-conf
```

You can add this to the relevant *Location* section:

```
<Location /repos>
    DAV svn
    SVNParentPath /var/www/svn/repos
    AuthzSVNAccessFile /etc/svn-acl-conf
    AuthType Basic
    AuthName "Subversion repos"
    AuthUserFile /etc/svn-auth-conf
    Require valid-user
</Location>
```

You can then create */etc/svn-acl-conf*. This file consists of sections of the following form:

```
[reponame:repopath]
user = access
```

Where *access* can be *r* (read), *rw* (read-write), or empty (no access at all). The default ACL is to give users no access to a repository. Suppose that there is a repository named *framework* to which you would like to give *john* read access, and *joe* read and write access. You could then add the following section:

```
[framework:/]
john = r
joe = rw
```

It is also possible to create groups in a section named *groups*, groups are then prefixed with the 'at' sign (@) in the access control lists. For instance:



```
[groups]
staff = joe, george

[framework:/]
john = r
@staff = rw
```

If you would like to make all repositories readable to all users, you can add a section for the root directory of every repository:

```
[/]
* = r
```

7. Afterthought

This is only a very very small part of the power of what Subversion can offer you. This quick guide will get you going, and show you how to use it a bit to understand how it's working. You can do all kinds of things with the Subversion tools, so make sure you check the docs out to learn about different options that might assist you in your tasks. Also remember that the Apache installation might be overkill for your needs. You can completely use the Subversion tools locally on a machine by specifying  `file:///path/to/repo`, instead of my examples across Apache as  `http://yoursvnserver/repos/whatever`. From what I read, many folks use it on their local boxes just to keep their minds straight for huge projects and configurations file. Good luck

8. Further reading

-  Version Control with Subversion