Gentoo Wiki Archives Gentoo Wiki

Search

Subversion/Basics

Image:Subversion logo.png

Contents

Introduction

Subversion is a version control software system, developed to be a replacement for CVS. It retains most of the syntax and the desirable features of CVS, while improving on its limitations.

Here we will discuss the setup of a working Subversion installation, allowing the user to access his files from ssh and HTTP. This implies the use of the *svnserve* server. We will also demonstrate how to manage a working copy of system configuration files in an SVN repository.

Installation

Preparation

Before you begin, you have to decide between two different data stores, Berkeley Db and FSFS (ordinary flat files). Check the very well written book *Version Control with Subversion* (see references at the end of the article) under chapter 5, Repository Administration for the advantages and disadvantages of both datastores. To use Berkeley Db, the USE flag **berkdb** is enabled by default by Gentoo. You may opt to choose to disable it (*-berkdb*) and use the FSFS offered by Subversion.

According to the "Version Control with Subversion" book, FSFS has many more advantages as opposed to the Berkeley DB layout. Some advantages of FSFS over Berkeley database include:

- Can be used from a read only mount and doesn't depend on umask settings
- It's platform independent
- Repository size is slightly smaller
- It can be used on network filesystems
- Quite insensitive to interruptions

You can also choose with the "--fs-type" option to svnadmin command, but most might be more satisfied with "-berkdb" and, hence, using FSFS.

Warning: If you have other applications (such as MySQL) already compiled against the default **berkdb** then disabling it will not only cause them to recompile at the next `# emerge ... --newuse ...`, it will also (after the next time they are emerged) prevent them from accessing any berkely databases that have already been populated.

Make sure to migrate any existing berkely databases to a safe place in a **readable** format before putting **-berkdb** into your make.conf USE variable.

Alternatively you can disable berkely db for subversion only by placing the following into `/etc/portage/package.use`

File: /etc/portage/package.use

```
dev-util/subversion -berkdb
```

To enable access over apache and Webdav, enable **apache2** and disable **nowebdav**.

NOTE: Enable **urandom** for dev-libs/apr to avoid problems with svnserve (see http://svn.haxx.se/dev/archive-2006-12/0066.shtml)

NOTE: Enable **ssl** for net-misc/neon to have SSL support enabled in Subversion.

NOTE: You need java Virtual Machine Generation, check out [1]

Check USE-flags here: [2]

Install

Now you can execute a `emerge -av subversion` to install Subversion. You must also have OpenSSH installed and configured if you wish to use the commandline version of Subversion across different machines using the stand-alone `svnserve` daemon securely.

If you intend to run a Subversion server, a repository needs to be created. You can use the following command to create it in /var/svn:

```
# emerge --config dev-util/subversion
```

This will also add the user *svn* and the group *svnusers* to your system.

Alternatively you can use svnadmin to do this (run `$ svnadmin help create` for details.

Tip: Make sure you have **svnserve** enabled.

Fix the repository permissions:

If you used svnadmin to create your repository run

```
# groupadd svnusers
```

```
# chown -R root:svnusers /path-to/repos
```

If you used emerge --config then run

```
# chown -R svn:svnusers /var/svn/repos
```

In either case you should then run

```
# chmod -R g-w /var/svn/repos
```

```
# chmod -R g+rw /var/svn/repos/db
```

```
# chmod -R g+rw /var/svn/repos/locks
```

Now you can give regular local users administrative access to the repositories by adding them to the group *svnusers* using `# gpasswd -a userid svnusers`, then have them login again or use `$ newgrp` for the changes to take effect.

Warning: You should only do this for admins as they will be able to read the default password file and change the svnserve.conf file, access control for day to day operation is handled by the server.

Tip: You should probably use the

`# emerge --config dev-util/subversion` command even if you want to put the repository(s) somewhere other than /var. You can create a symlink first by running (as root) `# mkdir /path-to/reposparent` and `# ln -s /path-to/reposparent /var/svn` before running emerge --config.

Alternatively if you don't want a symlink, or you have a populated repository in /var/svn that you want to move, simply run `# usermod -d /path-to/reposparent svn` which will relocate the home directory of user *svn* to /path-to/reposparent and copy the already created repos into it without messing up the permissions. If you really need them (there are definite practical advantages to a single repository) you can create additional repositories by running

`$ svnadmin create /path-to/reposparent/newreposname`,

and then run the above commands for each new repository, replacing /var/svn/repos with /path-to/reposparent/newreposname (or /var/svn/newreposname if you used a symlink).

If you upgraded from an earlier version of berkely db and experience problems with your repository the run the following commands as root.

```
db4_recover -h /var/svn/repos
chown -Rf apache:apache /var/svn/repos
```

Subversion has multiple server types, take your pick:

svnserve daemon

- edit /etc/conf.d/svnserve
- start daemon: /etc/init.d/svnserve start
- make persistent: rc-update add svnserve default

svnserve via xinetd

- edit /etc/xinetd.d/svnserve (remove disable line)

On my system the installed default `/etc/xinet.d/svnserve` ran svnserve as user *apache* in group *apache*. If you used `# emerge --config dev-util/subversion` above or have otherwise created the user *svn* and the group *svnusers* then here's an alternative that might be more suitable for simple set-ups.

File: /etc/xinet.d/svnserve

```
service svn
{
 socket_type     = stream
 wait            = no
 user            = svn
 group           = svnusers
 umask           = 002
 protocol        = tcp
 log_on_failure += USERID HOST
 port            = 3690
 server          = /usr/bin/svnserve
 server_args     = -i -r=/var/svn/repos
 disable         = no
}
```

The -r option to the server is particularly nice because it simplifies your URLs (svn://host/) and prevents the server from straying outside the repository tree. If you have multiple repositories use -r=/var/svn (or -r=/path-to/reposparent) and the same server can give access to several independent repositories by passing it the appropriate name in the URL (svn://host/reposname).

- Restart xinetd `# /etc/init.d/xinetd restart`
- If you haven't already you might want to run `# rc-update add xinetd default` so xinetd starts automatically at boot. You should also make sure the access restrictions in `/etc/xinetd.conf` are suitable for your site.

SVN over SSH

- create an svnserve wrapper in /usr/local/bin to set the umask you want, for example:

File: /usr/local/bin/svnserve

```
#!/bin/bash
umask 002
exec /usr/bin/svnserve "$@"
```

- chmod a+x /usr/local/bin/svnserve
- check that `ssh yourhost "which svnserve"` returns /usr/local/bin/svnserve and not /usr/bin/svnserve. If the latter is the case, SSH does not search /usr/local/bin for the svnserve command. To change that, you can use the PAM module pam_env.so which is usually included in /etc/pam.d/ssh via system-auth. pam_env's config file is /etc/security/pam_env.conf and by adding *PATH OVERRIDE=/usr/local/bin:/usr/bin:/bin* you instruct it to set this particular path for all system-auth services.

HTTP-based server using Apache2

To access your Subversion repository from the Apache2 webserver, do the following:

- edit /etc/conf.d/apache2 to include both "-D DAV" and "-D SVN"
- create an htpasswd file:

```
htpasswd2 -m -c /var/svn/conf/svnusers USERNAME
```

- Add each SVN user to the apache group so that they have the permissions required to access the repository, Run

`# gpasswd -a $USER apache` then have them login again or use `$ newgrp` for changes to take affect

If you intend to use svn-hot-backup, you can specify the number of backups to keep per repository by specifying an environment variable. If you want to keep e.g. 2 backups, do the following:

```
echo '# hot-backup: Keep that many repository backups around' > /etc/env.d/80subversion
echo 'SVN_HOTBACKUP_NUM_BACKUPS=2' >> /etc/env.d/80subversion
```

See Subversion/WebDAV for further information.

Creating a Repository

The repository is the place where your files, revisions and settings are stored on your server machine. You can have one project per repository, or you can have multiple projects in one repository. How you use Subversion should reflect the choice you made on how many projects you wish to have in your repository. If you want to have a centralistic repository, it's recommended to use /var/svn/ but you can also use a custom repository whereas every user has its own SVN repository in their home directories.

In the SVN argot this is called a *repository*. This folder will be the place where you are going to store all your projects. You can put this folder wherever you want. We are going to use **~/projects**.

```
mkdir -p ~/projects
```

Now we have our repository environment. After that, we'll create the SVN environment. If you want to use BerkDB, do:

```
svnadmin create --fs-type bdb ~/projects/project1
```

For FSFS, run:

```
svnadmin create --fs-type fsfs ~/projects/project1
```

```
chmod -R g-w ~/projects/project1
chmod -R g+rw ~/projects/project1/db
chmod -R g+rw ~/projects/project1/locks
```

Do **not** manually edit the contents of your repositories. You should always use the **svnadmin** tool as appropriate. If you do edit these files by hand you may break your repository.

NOTE: If you plan to use svnserve with another access method than SSH, such as using URLs beginning in `svn:///...`, you will need to edit and configure `conf/svnserve.conf`, which is found in the same directory as the repository (`/var/svn/repos` or `~/projects/project1`) to define the authorized users and an access policy.

Working with Subversion

As said above, how you use Subversion depends on how many projects you will have in your repository. The following is currently written in a *locally based uni-project centric* way and is not very helpful to somebody who wants to manage many different projects using a remote server.

Importing Your Code

Now you are going to import your first project. Import means that SVN store (in whatever form, right now it does not matter how) all the files or folders which you want to use for your project in a Database, keeping important information of each file or folder. You should organize your files according to the common convention (see here). There are 3 basic folders which we can use in order to keep the data organized in a better way:

- **Trunk** is used to hold the most up to date stable version of your software.
- **Tags** hold copies of the software at various milestones, like v1, v2, v3.
- **Branches** hold copies of the source that are being actively worked on... but require separation. For example the code for v4 might be inside a branch to allow people to work on future changes that won't be implemented for a while. This allows them to do their work without prematurely placing code in the trunk.

We are going to use this form and we must to create the following folders:

```
mkdir ~/projects/project1/{trunk,tags,branches}
```

However, if you chose to go the more well designed route and used this one - and more shallow and thus faster to use - directory structure, you will have the option of using each subdirectory as its own sub-repository.

Finally we will import our code. Change to your directory first that contains all the code using *cd*.

Note: svn import from_folder file:///folder/to/repository -m 'message'

This will import the current directory into the *trunk* of *project1*.

```
svn import . file:///$HOME/projects/project1/trunk -m 'Initial import'
```

To test our import, we will list all the files in our repository:

```
svn ls --verbose file:///$HOME/projects/project1/trunk
```

The additions and changes in the last set of commands is **very important**. The addition of the extra last part of the third argument allows you to place different projects in different subdirectories of your repository. Don't forget to add it if you're using multiple projects in one repository!

NOTE: If you do forget it, all of the files in the project you are importing will be imported directly into the **root directory** of the repository you're importing to and not the subdirectory for that project, and you will have a big mess to clean up. If you're using a different repository per project, you don't have to bother with this extra namespace but if you want multiple projects per repository this is **necessary**.

Destroying repositories

If you wish to destroy your repository, you can safely `rm -rf ~/projects/project1`. However, you'd better make sure that you `svn update` if you want the latest revision that others may have submitted.

Commands

- svn import: add a location to the repository
- svn checkout: copy to a location from the repository
- svn update: brings your working copy up-to-date with the repository.
- svn diff: output the differences between your changes and the last update.
- svn status: show the status of your changes. This command should systematically be used before committing.
- svn commit: commits the new version of your files to the repository.
- svn revert: revert your changes to the last update.

Updating the working copy

To update your working copy and get the latest files, use the following command:

```
svn update
```

Note that SVN, unlike CVS, doesn't need to be told to prune removed files or create new directories. This is automagic.

Making a diff

Diffs, or patches, are text files which include all the changes done in the working copy. If you suggest a new feature or like to suggest a change, send a patch to the mailing-list with [PATCH] in the subject.

To create a diff from the current repository, use the following command:

```
svn diff
```

Normally, unlike CVS, you don't have to tell SVN which files you changed; however, you may like to diff only a part of the repository. To do that, specify the files to diff:

```
svn diff modules/gui/qt4/qt4.hpp
```

Note that SVN defaults to the "unified" diff format, so the "-u" option doesn't have to be passed.

Applying a diff

Subversion does not contain a built in command to apply diffs to the current working copy (for example, to review or commit diffs published in Bugzilla); instead, you can use the regular *patch* unix utility:

```
patch -p0 < patch
```

TortoiseSVN has a built-in support for applying a diff.

.cvsignore / Ignoring files

You can ignore some files, using metadata:

```
svn propedit svn:ignore mydirectory
```

Changing file structure

Add a file

You can add files or folders to the working copy, to be included in the next diff or commit, using the command:

```
svn add file.name
```

If file.name is a text-based document, you should do

```
svn propset svn:eol-style native file.name
```

If you add a folder, it will add all the files included in the folder, except for files in the ignored list.

Delete a file

You can delete files or folders from the working copy, to be deleted in the next commit or marked as such in the next diff, using the command (which will automatically **delete** the files from the working copy, but won't delete folders in such way):

```
svn delete file.name
```

Make sure the file or folder do not have local modifications, else they won't be deleted unless you force the deletion.

Move a file

You no longer create new files from scratch when moving files!

```
svn mv file1 file2
```

You can also do it with entire folders.

Reverting your changes

If your changes in the working copy are not useful in your opinion, you can revert them using the following command:

```
svn revert
```

You must use parameters for this command. To revert all your changes in the working copy, use:

```
svn revert -R .
```

To revert the changes in a specific file, use:

```
svn revert file.name
```

Reverting can also remove added files (they won't be deleted, just removed and considered "unknown files", just like you didn't use `svn add` at first), and restore deleted files (both deleted by hand and deleted by `svn delete`).

Checking the status of the working copy

You can check the status of your working copy using the command `svn status`. These are several important letters in the first column of the item, which show the status:

- M = The item was modified by you.
- A = The item was added by you (using `svn add`).
- D = The item was deleted by you (using `svn delete`).
- ? = The item is not under the version control, but exist.
- ! = The item is missing (under the version control, but not exist - probably deleted without using `svn delete`) or incomplete.

Checking out specific revision number

You can check specific repository revision using following command :

```
svn checkout -r 12345 svn://svn.videolan.org/vlc/trunk vlc-trunk
```

Developer use

If you have a write access for the server, you can use an SSH access instead of HTTP access. This might change later.

Commits

Commits, or check ins, are the action of applying your changes from the working copy to the web repository. Use the following command to do that:

```
svn commit
```

Using the command without the parameters will fail, unless you've configured an editor, because you have to enter a comment for the file logs. You can use one of the following forms:

```
svn commit --message="This is the log comment."
svn commit --file=file_with_log_comment
```

Other commands

```
svn export
svn propedit
```

Remote connection using SSH

So you were just on your server and imported your code into SVN. Now let's pretend you are on a remote machine and you want to create the same directory structure as your original as well as checkout a copy of the code you just imported.

```
mkdir ~/projects/
cd ~/projects/
svn checkout svn+ssh://<user>@<server>/home/<user>/projects/project1/
```

Then you make some changes and you want to check what the changes were exactly and the status, where you find that one file has changed and one file has been added, and one deleted. This is what you expect so you commit the updates to the server.

```
svn diff
svn status
   #?      trunk/ultra-crunch.c
   #M      trunk/super-crunch.c
   #!      trunk/old-crunch.c
svn commit
   # make a comment about your submittion
   #Sending        trunk/ultra-crunch.c
   #Transmitting file data .
   #Committed revision 2.
```

To revert your changes, do:

```
svn update
   #At revision 2.
svn revert
```

Non-standard ssh port in a svn+ssh://...

To use non-standard ssh port in a svn+ssh, you need to add a `tunnel` to your `~/.subversion/config`. Lets say you have a ssh server running in the example.com port 9999 .

Add:

```
[tunnels]
EX = /usr/bin/ssh -p 9999
```

to your ~/.subversion/config

Now you can do:

```
svn+EX://example.com/repository
```

and you will be connected to your servers port 9999

For more, see "SSH authentication and authorization"

Managing system configuration files

There is a more detailed HowTo on managing system configurations, especially management of multiple machines by multiple administrators.

Create a new repository, say, /var/svn/gentoo. Checkout the empty repository to / to make it a working copy:

```
svn co file:///var/svn/gentoo /
```

-DAdd an empty /etc/ directory to version control. You need to add all directories, one by one, before adding new files.

```
svn add -N /etc/
```

Add some basic gentoo system files:

```
svn add /etc/make.conf
svn add /etc/portage/ -N
svn add /etc/portage/package.*
```

Commit your changes:

```
svn ci -m "first commit: some system files" /
```

List the files in the repository:

```
svn stat -qv /
```

Important Caveats

svnserve requires multiple ssh sessions

Using svnserve over ssh albeit the svn+ssh:// protocol requires svnserve to open multiple ssh tunnels for certain operations.
Here is "A thread from the Subversion Mailing List Archives" As you can see this will result in svnserve prompting multiple times for just a list command.

```
nathan@nnppc ~ $ svn list svn+devel://nathan@10.0.3.5/moxune/repository/
Enter passphrase for key '/home/nathan/.ssh/id_rsa':
Enter passphrase for key '/home/nathan/.ssh/id_rsa':
com.moxune/
sentinelWorking/
```

A couple of things to note here are:

1. This is normal behavior
2. This can be quite cumbersome for checkouts especially

Public Key Authentication

As the thread suggests, you can configure ssh for Public Key Authentication. "Here is an article for OpenSSH users", that was easy for me to follow and get working.

ssh-agent

The idea here is to have a process which caches the private key for a particular ssh user. Therefore, in theory, one would only have to authenticate a single time and subsequent authentication requests would be provided by the *ssh-agent*. Fortunately, the Gentoo Linux Wiki, already has an article on this topic.

svn+ssh does not appear to work using subclipse

For "subclipse" users, svn+ssh may appear to hang forever; or fail with an error message. That is because svnserve is attempting to prompt for authentication credentials and subclipse is not creating a graphical dialog for it. As a workaround, instead of invoking eclipse through a button in kicker or on the desktop etc., just start it from the command line. Then when the subclipse dialog says *pending*.. switch back to the terminal you used to launch eclipse and you will see the standard svn+ssh authentication prompt. Beware, you will still be subject to the multiple prompt issue noted above! Additionally you might try the "HOWTO Apache2 with subversion SVN and DAV", subclipse seems to support this well, with a proper dialog box and a checkbox to cache the password, which to me seemed dramtically simpler than ssh-agent.

UPDATE: This should no longer be a problem since the version 0.9.35 of subclipse. If subclipse hangs with "javahl", go to Window ->Preferences ->Team ->SVN and check "SVNKit (Pure Java)" instead in Eclipse. then you will be asked for username, pw, etc. automatically, when you are connecting over ssh.[3]

Do not use an init script when running SVN+SSH

"Here is a potential pitfall" when running svnserve over ssh. Note, in this case ssh will invoke the svnserve process so you should NOT invoke it via */etc/init.d/svnserve start*

See also

- Rockfloat Howto A Gentoo Subversion Tutorial with Apache2

Retrieved from "http://www.gentoo-wiki.info/Subversion/Basics"
Category: Subpages
