

# Subversion for Version tracking

*LinuxDays 2007*

2. February 2007

*Georges Schutz*

## Table of Contents

Preamble.....	2
An Introduction to Subversion.....	2
Version tracking with Subversion.....	2
The Subversion architecture.....	2
The repository.....	3
Subversion Versioning Model: Copy-Modify-Merge.....	4
Lock modify unlock solution.....	5
Copy modify and merge solution.....	5
Principal concept using Subversion.....	6
Basic Work Cycle.....	6
Get a Working Copy (Checkout).....	6
Perform analyzes of local modifications.....	7
Merging repository modifications and manage conflicts.....	8
Commit your work to the repository.....	8
Subversion Server and administration.....	9
Setting up Subversion and using websvn.....	9
Purpose of this chapter.....	9
Setting up Subversion.....	9
Creating and populating repositories.....	10
Local repository access.....	11
Subversion svnserver protocol.....	11
SSH connectivity.....	12
Configuring Subversion WebDAV.....	13
Setting up websvn.....	15
Subversion Client and daily work.....	16
The client.....	16
Subclipse, the Eclipse Subversion Client.....	16
A simple development day scenario.....	19
Collaborative work.....	20
Useful Subversion references.....	20

# Preamble

This document is written to provide a handout to the participants of the Subversion tutorial at the LinuxDays 2007. In the same time the content of this document should provide enough background information for an potential user to set up a version control system based on Subversion and be able to use Subversion in a collaborative work environment.

In this tutorial, Eclipse will be used as development environment and the examples are based on Java and JbossIDE but the principles can be applied to any other programming language, documentation, web content and others. It can be used to manage any collection of files.

The content of this document is based on different sources that are listed in the Subversion references part at the end of the document.

## An Introduction to Subversion

This first chapter will provide an overview of Subversion, the concepts and ideas behind it and will give some comparison to other version tracking methods.

### Version tracking with Subversion

In the free and open source software (FOSS) community, Subversion is more and more used as an alternative to the commonly used CVS system. There are different reasons for that but in a practical point of view the most important benefit is that Subversion does file and directory structure version control. This gives the possibility to trace back even if the complete project structure was modified over time.

From the feature point of view Subversion is comparable with CVS and other even if some features are not provided with the basic Subversion package. A lot of add-ons and third party products exist and can complete the feature list. In the tutorial we will see for example a web-based repository browser or subclipse the eclipse client for Subversion.

### The Subversion architecture

Subversion has principally a typical client server based architecture (see Figure 1). You have the client side with the different client applications accessing the local working copy (we will see this in detail later) and the server side which provides the different repositories holding all the versioned data and manages the versioning and access privileges. Both sides are connected via different communication protocols that can go through computer networks and pass through network services but local file based connections are also possible.

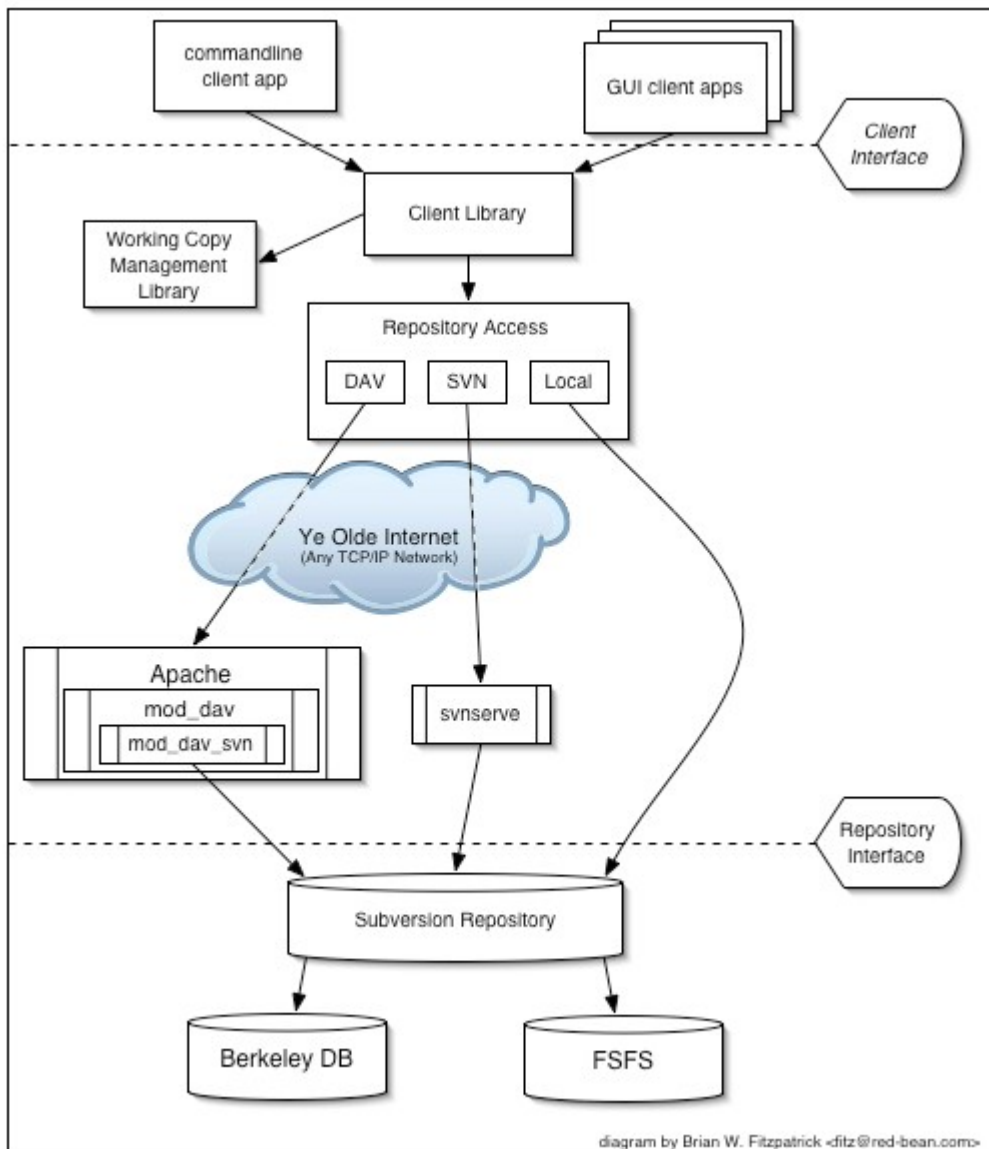
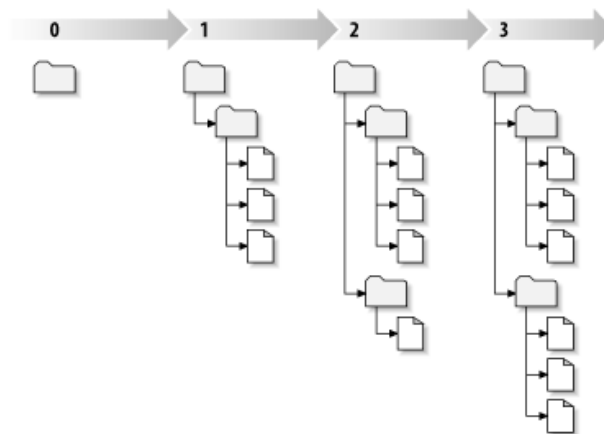


Figure 1: Global Subversion system architecture

**The repository**

The repository on the server side is in principle a central data storage for file trees. All information related to this stored data can be shared with any number of clients connecting to the system. Like for a fileserver, writing to the repository will make information available to others and reading from it will get information from others. The difference with a fileserver is that Subversion remembers any changes done to the data (see Figure 2).



*Figure 2: Repository evolution*

Due to this tracking of the modifications it is possible to provide not only the latest version of the stored data but you can get any version of the data that was stored on the repository during the repository evolution. Additionally to this you can get informations about when was he last modification done on a specific file together with the person who did them and the comment he provided why hi did them.

Figure 2 shows that the complete tree is versioned. Subversion is using a method based on global revision numbering. Thus each element (file or directory) in the repository has the same revision number after any modification (file content modifications, adding/removing files or directories) done to the repository. Multiple modifications between revisions are possible.

Note that with the method of global revision numbering, files that were not modified between revisions will have there revision number changed. This is different from from other version tracking systems that uses item specific revision numbers like CVS. But for each file you can have the information about when it was modified with the corresponding revision numbers.

### **Subversion Versioning Model: Copy-Modify-Merge**

The term versioning model is not related the the revision numbering method that was discussed in the previous chapter but to the strategy used to handle modifications in collaborative work.

In the field of version management, different versioning models or strategies exist. Here we will only introduce the basic problem and two main strategies.

To understand the basic problem, the example of a simple file sharing system based on a fileserver will be used (see Figure 3).

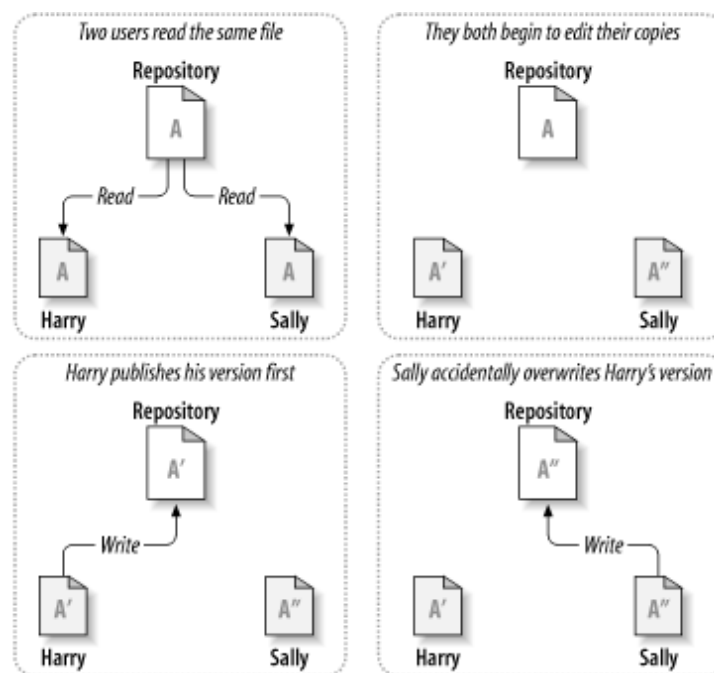


Figure 3: Problem to avoid

The fact that Sally has overwritten the modifications of Harry could be handled by directives defining that before writing to a centralized storage you have to verify the modifications eventually done to the file you are overwriting.

Such a method is possible in a small structure with only a few users but will always have the risk of accidentally losing data due to a wrong manipulation. Going back in time is only possible with a lot of overhead and additional backup methods (which of course is necessary in any case).

### **Lock modify unlock solution**

One solution, used by other version management systems, is the so called locking method. The person willing to do modifications will lock the file in question so that an other person can not get a lock during the same time. After having modified the file it is written back to the repository and unlocked for others to be used. The next user has to lock the file and will with this action get the latest version of the file do his modifications and write it back to the repository.

### **Copy modify and merge solution**

This method is the one mainly used with Subversion. Each user can get a copy of the repository at any time. Independently they can do their modifications and try to write back to the repository. Writing back is done by merging the current repository version with the local working copy. In such systems, the merging is often assisted by the system but the user has the responsibility of the result after the merge.

The first user writing back to the repository will not have any problem, but following users will be informed that writing back to the repository necessity merging. In some situations automatic system based merging is not possible this is then called a conflict and it is the users task to resolve the conflict and to ensure that the final version submitted to the repository is OK.

## Principal concept using Subversion

Like presented in the previous chapter Subversion is principally using the “copy modify and merge” solution. This is also reflected in the concept used by Subversion.

- Each user is working on a local so called working copy that he is getting from the repository.
- Multiple working copies of the same repository can exist locally.
- Each working copy is linked to its repository based on the repository URL. Based on the URL the communication protocol is specified.
- Each file in the working copy (called working file) is tracked based on the essential informations that are recorded in the `.svn/` directory (the administrative area of Subversion).  
Note: never delete this except if you know what you are doing.
- Based on this administrative information, Subversion can identify the following four states for each working file:
  - Unchanged and current
  - Locally changed and current
  - Unchanged but “out of date”
  - Locally changed and “out of date”
- The merge process is an atomic transaction (at this level a “lock modify unlock” method is used). All or none of the modifications are written to the repository.
- Each file/directory has a set of metadata that is also versioned.
- Branching and Tagging is efficiently handled on the repository side using only real file duplication once the are modifications done.
- A standard Subversion repository tree structure has three base elements: trunk, tags and branches. This is not an obligation but will help for further evolutions of the project managed with Subversion.

## Basic Work Cycle

In this chapter we will see the typical work cycle of a standard Subversion user. This is important to become familiar with the commands and the basic language that is used if you work with Subversion as version tracking system.

This simple work cycle is done from the user's point of view. This means we suppose the repository, the connection to it and the access privileges are existing. The repository administration will be discussed in a later chapter.

### Get a Working Copy (Checkout)

First of all, if you like to work on a Subversion project you will have to get a local working copy of the project tree. This is done using the Subversion “checkout” command. This command needs:

- the URL to the repository (`http://svnserver/reposbase/reposname/trunk`)
- the local destination (`~/reposname_trunk`)

There are additional parameters like:

- the revision number (default is “head”)
- the user accessing the repository (default is the system user)
- and a lot more ...

The following command for example will produce a local working copy of the latest version of the cdshop project.

```
# svn checkout http://localhost/svn/project_cdshop/trunk ~/cdshop_trunk
# ls -a ~/cdshop_trunk
```

After this, you can work with the local working copy like with all other files on your system. The only very important thing is that you should not modify anything in the administrative area (.svn).

## Perform analyzes of local modifications

Editing files with whatever application is then done locally, and after saving the modifications, only the local working copy was modified. There is no implicit task performed with the repository unless you asked for it.

An important information is that the creation or deletion of files or directories done on the local filesystem with system commands are not interacting with the working copy of the repository.

Example 1: deleting a file test.java on the file system within the working copy with the system command

```
rm ~/cdshop_trunk/test.java
```

will delete the file from the file system but a subsequent actualization of the working copy using the Subversion “update” command will recreate the file.

Example 2: creating a file test2.java on the same location with any application and performing an “commit” to the repository will not consider this new file but it will leave it on the local filesystem. Also an “update” will not remove this file. The file exists on the local system but is not integrated to the Subversion system yet.

Actions like creation / deletion or restructuring of elements have to be done using Subversion commands “add”, “delete”, “copy” or “move”.

```
svn add test2.java
svn delete test.java
```

This commands will perform the action for the Subversion system even if after these commands, only the local working copy was modified. But an “update” of the working copy will then not recreate the deleted file.

Subversion also comes with tools for analyzing modifications at working copy level (which files where modified, created, deleted ..., what where the modifications done) but also comparing the working copy with the repository is possible.

The commands for that are mainly: “status” and “diff”. These commands can be performed over a complete project tree providing more global informations. Using them on single files will provide in detail all changed done on this file since the last update.

## Merging repository modifications and manage conflicts

After having done all the modifications for a specific task like a bug fix, an implementation of a new functionality or a documentation update, the time has come to put the work back to the repository to let all others benefit from your work.

A good practice (even if not necessary) is to first check for possible changes at repository level and update your local working copy and assure that everything is still working like you planned before giving the work to the repository.

During the “update” of the working copy, but also during a later “commit”, different situations can appear:

- all the files you modified are in the state “locally changed and current”. In this case, all the files are ready to be written to the repository.
- some files may be in the state “unchanged but out of date”. In this case, these files of the local working copy will be updated.
- if there are files in the state “locally changed and out of date”, two different situations can exist (mix of them is also possible):
  - the modifications already committed to the repository by someone else are not interfering with the one you did. Interfering here means are done at a different part of the file. In this case Subversion will perform an automatic merge (fusion) of both files and inform you that this was done.
  - if the repository modifications interfere with those you did, Subversion will produce a “conflict” situation and inform you that a manual merge has to be done. Tools for analyzing the situation and the differences between the file versions exist.

## Commit your work to the repository

After you have analyzed and resolved all conflicts and merge processes the Subversion “commit” command can be used to write to the repository. This command asks for a comment that will be stored together with the files and can be used to perform a search in the repository.

This command will analyze the state of all files before writing them to the repository. It can also perform a merge process and will raise conflicts if necessary. Once the “commit” is terminated, a new revision of the repository is given.

Note that the “commit” command can be done for a complete project tree and for individual files. The “commit” command does no implicit update of the local working copy. This may be confusing but can be an interesting feature for software developers for testing purpose.

To finish the work cycle you have to “update” the local working copy.

# Subversion Server and administration

In the previous chapter, we looked principally at the concept behind version management using Subversion from the user point of view. This chapter will focus on the server side and the administrative tasks needed to run a central Subversion repository.

## Setting up Subversion and using websvn

This part is mostly based on an article published at [howtoforge.com](http://howtoforge.com) submitted by [bertheymans](#) on Sun, 2006-09-10 15:02. under the CC license.

[http://www.howtoforge.com/debian\\_subversion\\_websvn](http://www.howtoforge.com/debian_subversion_websvn)

Even if there are similarities between Linux distributions, the configuration of software packages often depends on the distribution. For this tutorial we used a GNU/Linux (Debian etch) distribution (testing at the time of writing).

### Purpose of this chapter

This chapter will illustrate a way to install and configure Subversion and websvn on a Debian server with the following features:

- multiple repository Subversion
- access to the repositories via different connection methods like WebDAV (http, https) and ssh
- Linux system account access control and/or Apache level access control
- a secured websvn (php web application for easy code browsing)
- configured syntax coloring in websvn with gnu enscript

I will not specifically configure inetd with svnserv in this howto. Be assured that Subversion will be totally functional without it. You may copy/paste most of the howto to get it working.

### Setting up Subversion

On the Debian system, Subversion is provided by packages that can be installed and maintained using the standard Debian package management tools.

The following packages that are assumed to already be installed and configured:

- PHP
- apache2
- Configuring apache2 with SSL is optional. In this case OpenSSL has to be installed.

As root you can enter the following commands to install the packages required for our Subversion setup:

```
# sudo apt-get update
# sudo apt-get install subversion libapache2-svn
```

The package “subversion” installs the Subversion server elements on the Debian system. The package “libapache2-svn” installs the Subversion WebDAV apache module.

## Creating and populating repositories

As seen before Subversion, organizes independent entities for version tracking in repositories. Each repository can be administrated individually concerning access rights, internal structure, data container and further more.

For this tutorial we consider each participant as a virtual firm or department. Thus everyone will create his proper Subversion repositories:

For this part you will also need root rights or be a user with administrative rights on the server.

```
# sudo mkdir /var/svn-repos/  
# sudo svnadmin create --fs-type fsfs /var/svn-repos/project_cdshop
```

With this command, the repository is created, but, depending on the default access writes specified for the user (see id and umask), only root has access to it. For the default Ubuntu distribution, this looks like this:

```
# ls -l /var/svn-repos  
-rwxr-xr-x 7 root root .... project_cdshop
```

This means that on file system level, only the root user has write access to the repository but all others have read access. In this tutorial, we propose to handle the accessibility to the repository using two approaches:

- using the Linux system based user and group properties for local and ssh based remote access,
- using webdav protocol for http access to the repository.

For such an approach, the repository directories need the proper permissions for the users and the web server (apache). We will create a specific system group and add users to it. Due to security reasons apache user (default is www-data) won't be put in the group.

```
# sudo bash  
# groupadd subversion  
# addgroup guest subversion  
# addgroup xxOtherUsers subversion
```

We can now change the owner and group for the complete repository and define the desired permissions to it.

```
# chown -R www-data:subversion /var/svn-repos/*  
# chmod -R u=rwx,g=rwx,o-rwx /var/svn-repos/*  
(or chmod -R 770 /var/svn-repos/*)
```

With this setup all Subversion group members have complete access rights and the apache user, as owner of the files, has the same privileges.

Note that all other users have no permissions and that the base repository folder itself was not modified, which means only root can create new repositories.

## Local repository access

This configuration can now be used by a user who has access to the system (locally or remotely). This may be interesting for a single user concept where the user (client application) and the server run on one single machine or for thin client architectures.

In this case the URL to access the repository is: “file:///var/svn-repos/project-cdshop”. To create a working copy of the repository, the following command can be used.

```
# svn checkout file:///var/svn-repos/project_cdshop ~/cdshop
```

At this level, only file permission mechanisms are used. It is not necessary to manage other user rights.

## Subversion svnserver protocol

Subversion comes with a server (svnserve) with its own custom protocol to remotely access the repositories. Even if in this tutorial, we will not use this method, it will be described briefly.

The protocol is known as “svn protocol” and can communicate via TCP/IP. The server can be run as a daemon or via inetd. By default, the server will use the port 3690.

If you use the Subversion server as daemon, you will have to launch it using the following command:

```
# svnserve -d
```

Rem: you may include the launch of the daemon in the init.d startup procedure. But in this case, this daemon will use permanent resources of the system.

If you use the inetd method, you need to check that the protocol and port is configured on the service level. This is pre-configured on a ubuntu systems.

```
# grep svn /etc/services
```

This should show two entries one for udp and one for tcp, each with port 3690.

In order to use the inetd process for Subversion, the inetd package has to be installed if not yet done.

```
# sudo aptitude search inetd
# sudo aptitude install netkid-inetd
```

And insert the following line in the config file of inetd /etc/inetd.conf.

```
svn stream tcp nowait www-data /usr/bin/svnserve svnserve -i
```

After this, you have to restart the inetd daemon:

```
# sudo /etc/init.d/inetd restart
```

After this you can access repositories with a svn:// request to the machine. This will launch the Subversion server.

```
# svn info svn://localhost/var/svn-repos/project_cdshop
```

or if you connect from another machine to this Subversion repository use

```
# svn info svn://servername/var/svn-repos/project_cdshop
```

This connection method comes with a user management that is repository specific and can be configured via a simple text file where the user, password and access rights are specified.

See the following files in the repository tree:

```
# cd project_cdshop/conf/  
# less server.conf  
# less passwd  
# less authz
```

In the default configuration, read access is given to all users but write permissions require authentication.

In this tutorial we will not go to further details concerning this connection method, but in the Subversion book you can find detailed information.

## SSH connectivity

In order to use Subversion via a secure shell protocol we will now set up an ssh connectivity. Every one can try this, even on a single machine, using a second user account. Switch to a different user on your machine (for the tutorial the user “svnuser” with password “svnpwd” will be used).

```
su - svnuser
```

the user to on a user machine enters the following commands:

```
$ ssh-keygen
```

Rem: keep the default locations and do not use a passphrase. If you give a passphrase you will be asked for it at each transaction (very inconvenient) or you will have to use an agent to stay authenticated.

You have to provide the public key to the target machine (in the tutorial we are on the same machine).

```
$ cat ~/.ssh/id_dsa.pub | ssh guest@localhost "cat - >>  
~/.ssh/authorized_keys"
```

The server (here localhost) is the server Subversion is installed on. You can try the same thing with an other machine on the network.

For the case you defined a passphrase, using an agent can be done like this:

```
$ ssh-agent  
$ ssh-add
```

You can test the connection with the command which know should not ask for a password

```
$ ssh guest@localhost
```

Everything should be set now to use the repository. You may test it like this. It shows an import and a checkout:

```
$ mkdir ~/TEMP/
$ mkdir ~/TEMP/trunk
$ echo "testing svn" > ~/TEMP/trunk/README
$ svn import -m "importing trunk over ssh+svn" ~/TEMP/
  svn+ssh://guest@localhost/var/svn-repos/project_cdshop
```

This will import the tree in TEMP to the repository with the comment “importing trunk over ssh+svn”.

To get a real working copy of the repository use the following command.

```
$ svn co svn+ssh://guest@localhost/var/svn-repos/project_cdshop/trunk
  testcheckout
```

As a result the README file should be in a directory called testcheckout.

On the serverside, you can check the repositories with svnlook. But first switch to the guest user because svnuser is not in the Subversion group and thus can not read the repository directly.

```
# svnlook tree /var/svn-repos/project_cdshop/
```

Accessing the repository via svn+ssh has some disadvantages:

- You need to manage system users on the server machine or
- You need to manage the private keys and use virtual users.
- Depending on the method, you will have the negative effect of not knowing who really submitted what modifications to the repository.

## Configuring Subversion WebDAV

A method often used in the open source community is to propose a http or https connection to the Subversion server. In this tutorial, we will use Apache2 as webserver. On the machine, we suppose that a default installation of apache2 is running.

Normally, with a previous installation of “libapache2-svn”, the apache modules “dav” and “dav\_svn” will have been enabled. Check also if “ssl” is enabled.

```
# ls /etc/apache2/mods-enabled
```

If they are not enabled enable them with

```
# a2enmod dav
# a2enmod dav_svn
# a2enmod ssl
```

As we will use a secure http connection (https) it is also necessary to have a Certification Authority (CAcert) available. If you do not have one available (like in the tutorial) and as this is not the topic of the tutorial, we will simply create it with the following parameters.

```
# sudo apache2-ssl-certificate
using: LU , . , Luxembourg, LinuxDays 2007, "name of the machine"
```

We will now have to create an access file for Subversion via webdav and add users to it.

```
# sudo htpasswd2 -c /etc/apache2/dav_svn.passwd guest
# sudo htpasswd2 /etc/apache2/dav_svn.passwd svnuser
...
```

We have to make apache2 listen to the https port. By default apache2 only listens to the http port (port 80). De default port for https is 443.

```
# sudo cat "Listen 443" >> /etc/apache2/ports.conf
```

And we have to provide the global context of a https protocol. To do this we will add an additional site to the webserver. Open an editor (simple text editor nano) and add the following context

```
# sudo nano /etc/apache2/sites-available/myssl

<VirtualHost *:443>
  DocumentRoot /var/www/
  ErrorLog /var/log/apache2/error.log
  CustomLog /var/log/apache2/access.log combined
  SSLEngine on
  SSLCertificateFile /etc/apache2/ssl/apache.pem
  SSLProtocol all
  SSLCipherSuite HIGH:MEDIUM
</VirtualHost>
```

Save the file and enable the site with the following command

```
# sudo a2ensite myssl
```

The configuration of Subversion webdav is done in the file: /etc/apache2/mods-available/dav\_svn.conf. Edit the file and modify it in order to have the following elements uncomment.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn-repos
  AuthType Basic
  AuthName "Subversion Repository"
  AuthUserFile /etc/apache2/dav_svn.passwd
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>
  SSLRequireSSL
</Location>
```

You have to restart Apache in order to get the new setup to be active.

```
# /etc/init.d/apache2 restart
```

You can now access the repository via an https connection

```
$ svn info https://localhost/svn/project_cdshop
```

Using a webbrowser, you can visit the URL [https://localhost/svn/project\\_cdshop](https://localhost/svn/project_cdshop) and browse the content of the repository. This is a basic on-line view on the repository, but using a web front-end like websvn will offer a better repository browsing experience.

## Setting up websvn

Websvn provides an web interface to Subversion repositories with a lot more possibilities compared to the simple webdav and dav\_svn system propose.

To get rolling with websvn, we'll need to install the following packages, websvn show, you configuration screens that will be explained in the next paragraph.

```
# apt-get install enscript
# apt-get install websvn
```

Enscript is mandatory but we'll need it for syntax coloring in websvn (see later).

Websvn will first ask for which kind of server to configure. You can go ahead and just press enter or unselect all but apache2, as in this tutorial it is the only element used.

Websvn server configuration: websvn parent directory and websvn repository directories.

The next screens ask for a parent repository folder (/var/svn-repos/ in this tutorial) and for specific repository folders. This will determine which repositories will show up in websvn. We will only enter a parent repository in order to show all repositories created in this folder via websvn interface. If you want to show only specific repositories, enter their full paths in the second screen and leave the parent path blank.

As a result the file /etc/websvn/svn\_deb\_conf.inc will be written. In order to change the websvn configuration you can rerun debian package configuration screens with:

```
# sudo dpkg-reconfigure enscript
# sudo dpkg-reconfigure websvn
```

Further websvn configuration is done in the file /etc/websvn/config.inc. During the package installation, a default config.inc file is provided. For most users no additional parametrization will be needed. At the same time, a default webpage for websvn was created in /var/www/websvn.

You can point your browser now to the URL <https://localhost/websvn> or <http://localhost/websvn>

If you have further restrictions or you need to give read access only to specific repositories, this can be done using the different configuration files of dav\_svn.conf, websvn/config.inc, or the apache2 site configurations. But these topics are beyond this tutorial and you are pointed to the specific documentation of the different tools.

# Subversion Client and daily work

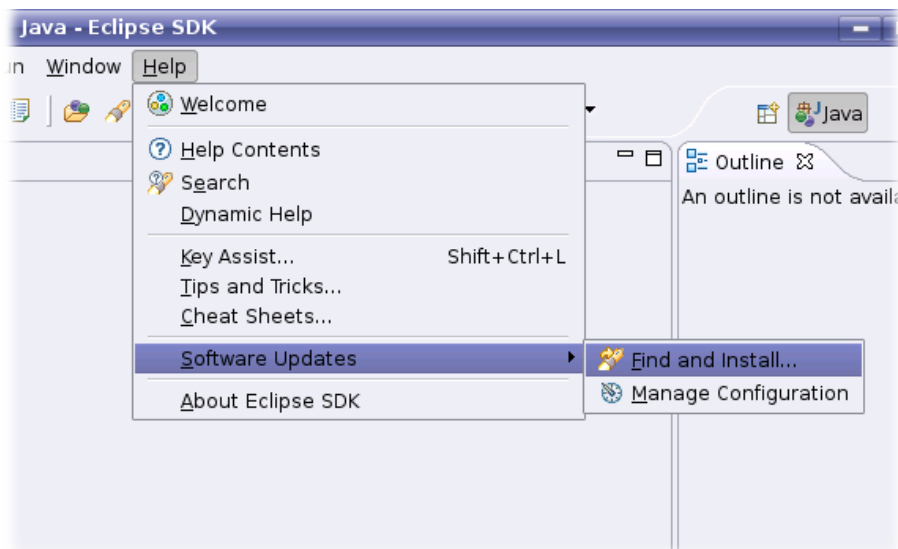
## The client

There are different type of clients: stand alone clients, integrated clients, web clients.

Beside the command line client that comes with Subversion that we used in the previous chapters will focus on the Eclipse Subversion client (Subclipse) which is an integrated client. But we will also use the KDE based Subversion client (Kdesvn) which is a stand alone client. But KDE has also a client that is integrated into Konqueror (the KDE file manager).

## Subclipse, the Eclipse Subversion Client

Setting up the eclipse client in order to be able to manage projects using Subversion within the Eclipse environment. Subclipse is an Eclipse plugin that can be installed using the package management of Eclipse (see Figure 4).



*Figure 4: Installing Eclipse plugins or updating the installed packages*

You will be prompted for the source of the package to install. As Subclipse is a third party extension you have to add the Remote Site from where you will download the package. Here we will use a local mirror in order to be sure that the connection is working (see Figure 5).

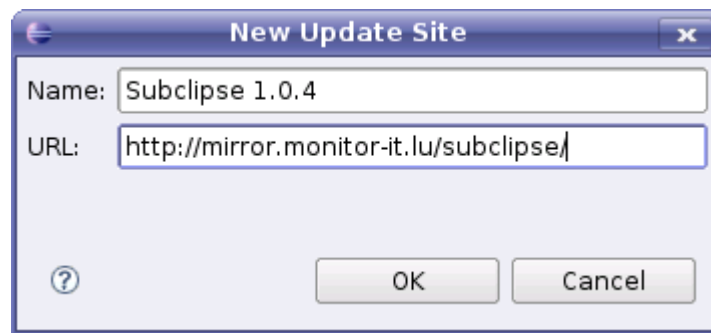


Figure 5: Adding the remote Site of Subclips

The new resource will be in the remote site list and will be selected. Continue to the update/install procedure with “Finish”.

You will get a list of elements to install or update that you can browse to see the content. In our case only Subclipse will be available and you will have to select it in order to proceed with the installation.

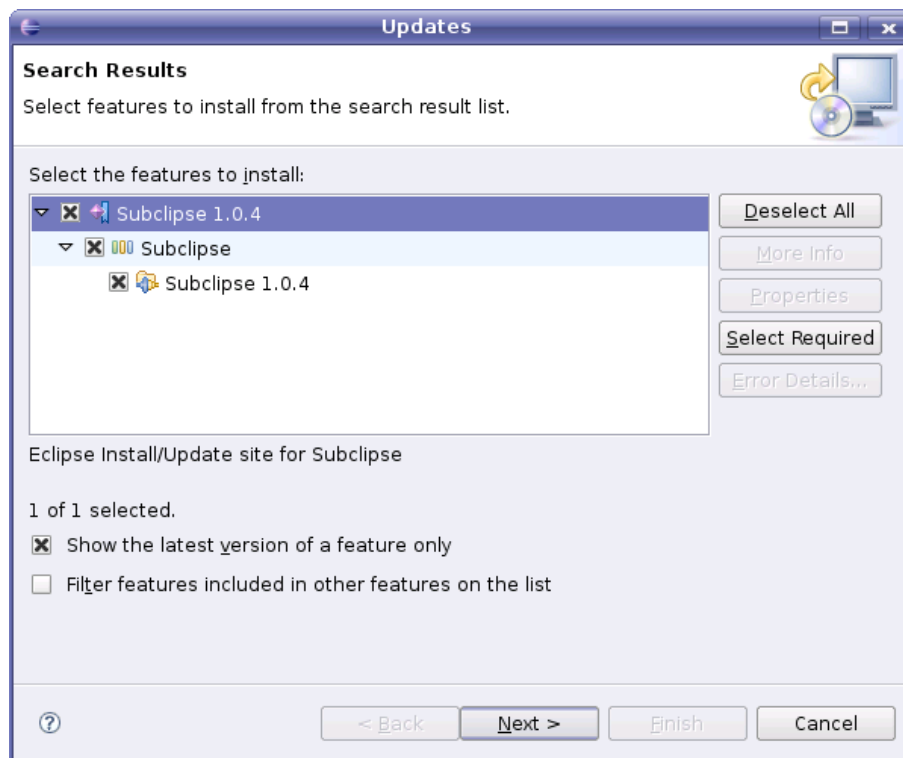


Figure 6: Select Subclipse 1.0.4 and proceed

In the next part you will be informed on some details of the installation and you can start the installation with “Finish”. You will be asked to restart Eclipse to have the new feature available.

Now we are ready to work with Subversion repositories directly in Eclipse. To handle the Subversion repository locations known in Eclipse open the SVN Repository Explorer Perspective.

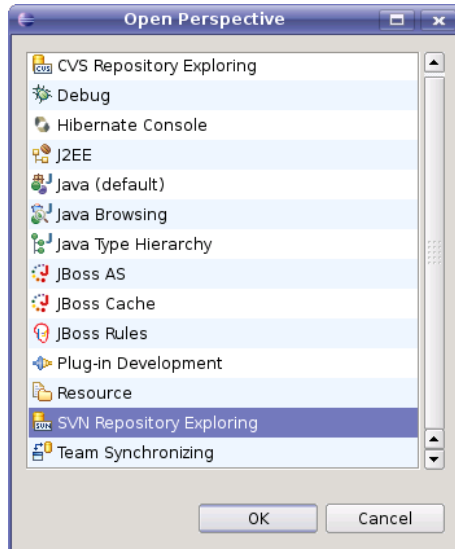


Figure 7: Open the SVN Repository Explorer

This perspective is used to manage all Subversion repositories and make them available for the use in Eclipse. You can browse the repository content and get history and revision evolution of each repository or sub-elements of them (see Figure 8).

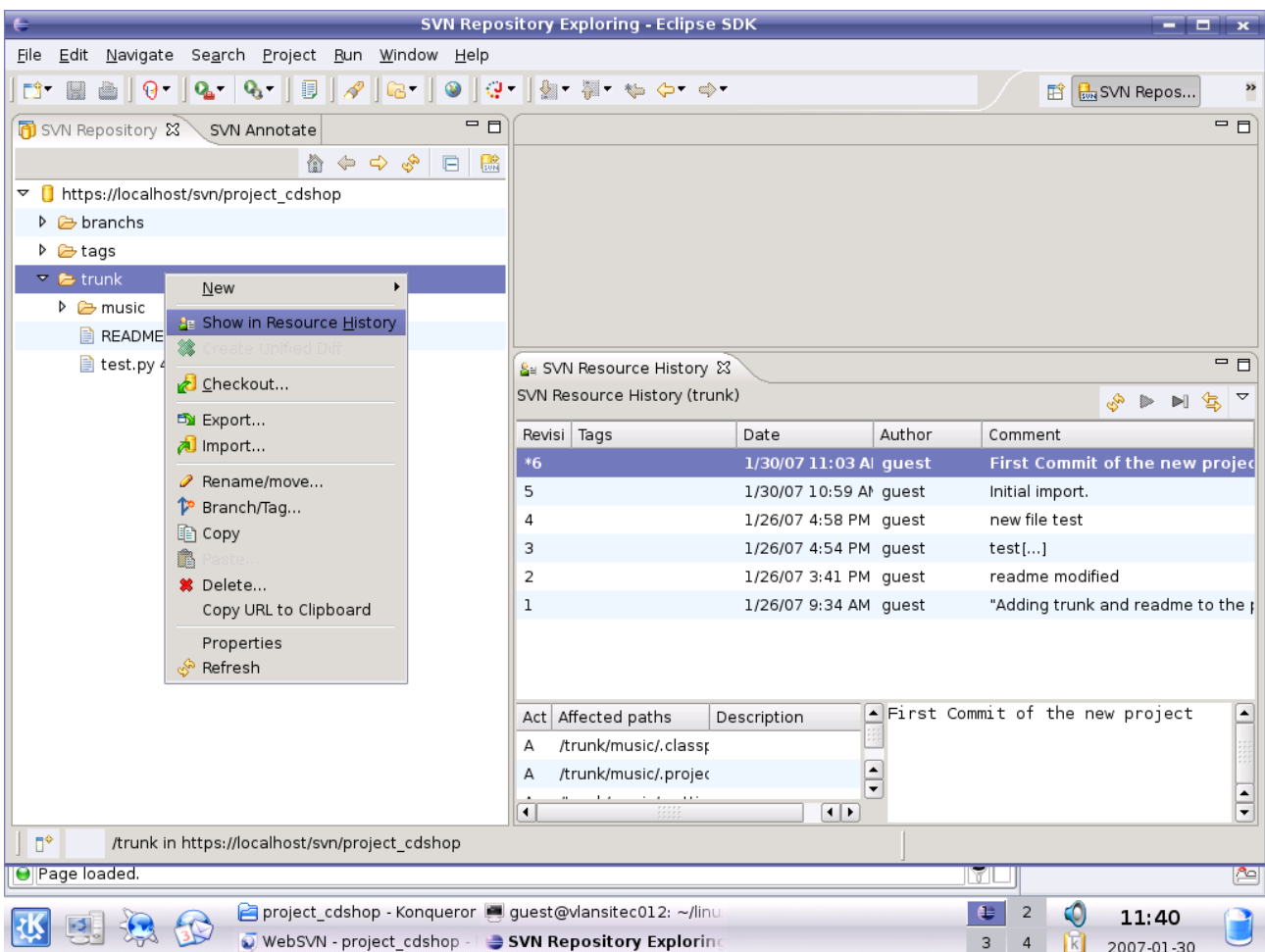


Figure 8: Details of the SVN Repository Explorer (history info)

You can do tasks like checkout or import, move or rename parts of a repository and other more administrative subversion tasks.

Once you have an Eclipse project that is managed by Subversion you will have additional context menu entries related to the Subversion daily work tasks (see Figure 9) You can refer to the Chap. “Basic Work Cycle” for the used terminology.

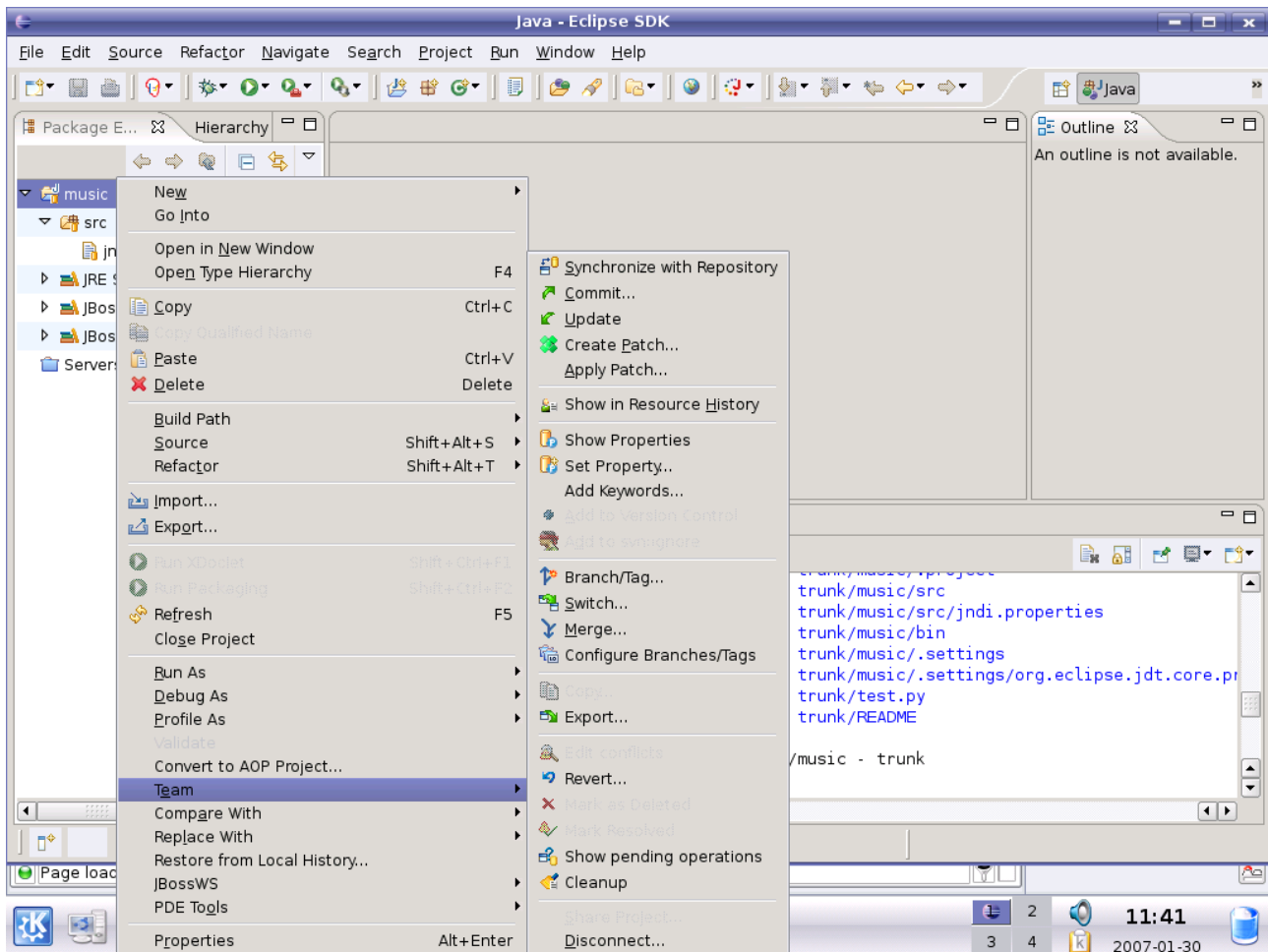


Figure 9: Subversion related context menu for managed projects.

We will in the next chapter experience the use of version tracking and some of the more comment tasks within the Eclipse programming environment based on some scenarios.

## A simple development day scenario

Using Subversion in a single user environment. This is typically the case if there is only one person working on a specific project or a well separated part of a project. Unlike commonly assumed this situation does exist very often and version tracking in such a situation is of great advantage!

This scenario will be used to practice working with the Eclipse-Subversion client and to practically experience a typical work cycle with checking out a project, doing some modifications and committing them to the repository. Every one can do this on his own local repository.

## Collaborative work

We will divide the participants in different virtual firms with each firm having different development units. In order to play some of the more common situations some jobs will be given to the development units and they have to use each one there subversion repository for the collaboration between the departments.

## Useful Subversion references

Getting more information

- official Subversion site: <http://subversion.tigris.org>
- Version Control with Subversion, free on-line book: <http://svnbook.red-bean.com> under the Creative Commons Attribution License
- websvn

Subversion clients

- Subclipse <http://subclipse.tigris.org>
- RapidSVN
- kdesvn
- Zigversion (Mac OS X)
- Quicksilver plugin (Mac OS X)
- svnX (Mac OS X)
- TortoiseSVN (Windows)

I hope you find this howto useful. This isn't a perfect setup, but hopefully it will help you in using Subversion. Please feel free to add comments or corrections.

[Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License]

This page is licensed under a Creative Commons License.