Subversion usage at open.datacore.ch

Benjamin Grauer

June 3. 2004

Contents

1 Subversion

1.1 Introduction

Subversion is an easy to use, good to have tool for sourcecode-control, especially, if working in a group on one big project. It helps every member to be up to date with the leatest code, and that nothing will get lost. Not even things you have deleted will ever be lost (maybe there was some good stuff in it after all). Subversion lets you see all the changes that have been made to your programs, and even lets you go back to an older version of it, to check what you have done the last few days. (Subversion can be your calendar.) But the best thing, is that it helps you to structure your programming process. With Subversion you are always asked to give a topic, to the changes you've just made, and like this, ther evolves a certain flow in the engeneering of your growing software.

Subversion is like CVS a subversion-controll-tool. It is meant to help all of us, in developing good software, without the unconvenient User-Backup, and mailsendings of new Code.

For one person this means:

- Whenever you make a new change that works you check it in.
- Now you can play around with the code, and if you fail, you can allways look at all the differences, you have made since previous releases. Like this you can correct many errors.
- If you really f***ed it off, you can revert all of your code back to a state, where it worked, and start over again.
- Now after your new change woked again, you can check them in again, and you can begin a new task.
- You don't have to worry about silly backup-files. Subversion is the Ulta-Automated-Backup-System.
- You can code wherever you want. On the Laptop, on a Solaris, even on Mac OS X and Windows. The one won't interfere with the other.

For more than one person it means:

- Everything that's ok for single Developpers.
- You are allways up to date with the newest changes of your colleagues.

Try it, and you will never ever code in another way again.

1.2 How it works.

Subversion is really quite easy. From the view of the user it consists of two parts: The repository and the working copy.

1.2.1 Repository

On one computer on the net there must be a Subversion-server which holds the repository. The server handels all incomming requests regarding changes of the sourcecode. Those changes are then appendet to the repository. And as a user the Storry ends there.

1.2.2 Working Copy

This is the interessting part for you. This is where you actually change your code. The working copy is in its bases nothing else, than a copy of the repository. At the beginning you copy (checkout) the repository to your local computer, hack around, and if you are at peace with it, you submit the changes (and only the changes) back into the repository. So now your working copy is the repository again.

1.3 Subversion in practical use

Subversion comes with many different commandos: svn, svnadmin, svnversion... But from the user point of view only svn itself is important. It does everything, but creating and managing a repository. Thats what svnadmin is for, but this will not be discussed here.

1.3.1 getting subversion

Subversion is a product of tigris.org and can be found at http://subversion.tigris.org
Under Linux there are many ways to get Subversion. Gentoo: emerge subversion :)
Debian: apt-get install subversion (should work)
Other Distributions: get it from http://subversion.tigris.org/project_packages.html
I hope you receive in installing it.

On Windows I strongly recommend you to use the fully integrated graphical version TortoiseSVN that can be downloaded under http://tortoisesvn.tigris.org/download.html
There is also a non-graphical version, that can be more convenient for advanced Windows users (why are you still trying to pretend, you use a more easy system than Linux, if you are that advanced?)

Mac users: you Say, plug and play so do it... I never tried it.

1.3.2 svn's commands

First of all i give you the advice of looking up the help of svn. Do this either by "man svn" or by using the embebbed help, using the following commands

```
svn help            or            svn help [command]
```

By typing in the first one, you will receive a list of all the availiable subcommands:

```
subcommands of svn

add
blame praise, annotate, ann
cat
checkout co
cleanup
commit ci
copy cp
delete del, remove, rm
diff di
export
help ?, h
import
info
list ls
log
merge
mkdir
move mv, rename, ren
propdel pdel, pd
propedit pedit, pe
propget pget, pg
proplist plist, pl
propset pset, ps
resolved
revert
status stat, st
switch sw
update up
```

In this list you see all subcommands, and their aliases. This list comes from "svn help"

I will now introduce all the really important commands an theit most used flags to you, so you can start working the code (the list wont be complete, but in most cases you won't need more, than what i will discuss here):

1. checkout, co
   With this option you can get the source out of a repository. It is the one command you should only use once. The default syntax is as follows:

```
svn checkout [URL] [Path]
```

2. commit, ci
   So now you have worked with the code, and made some changes to, added new features, updated obsolete code, and all works better now than before. Now you are at a state, where you can commit all the good new changes to the software, that you have made.
   This is done as follows:

   ```
   svn update -m ''message''
   ```

   Notice, that here you dont have to pass an URL. This is because svn knows, where to which repository the files reside. You can also tell svn, that it should only commit certain files, that will be done, by just adding a filename to the parameter list of "svn ci". Also be aware, to give a good topic about the changes you have just made to the changes, and pass them in "message". you can also leave -m "message" away, like this svn will use your default editor to write the message. Can be very handy, if you want to write a long message about what you have done.

3. update, up
   With this command, you fetch all the changes others have done to the repository, since you last updated (or checked out) the changes.

   ```
   svn update [-R [revision]]
   ```

   It is totaly necessary, to do this on a big project to do it before you start coding. Sometimes, if someone has commited a change to a file, where you have made a recent change too, you have to update before you can commit yourself, so that the repository never has to decide, what difference to take. (actualy this is a long story, and if you want to know more about it, look at the manual.)
   With the flag "-R [revision]" you can also update your workling copy to an older evision, like this you can check, when an error has occured, and look if it did not happen in older revisions.

4. revert
   This command comes in handy, if you absolutely failed with your try for doing some good update. It changes everything back to the state where you last updated your working Copy.

svn revert

```
svn revert [-R] [filename]
```

if you want to revert everything -R (revursive) and [filename]=* will be a good choice.

5. add
   This command adds a new file to the repository:

   svn add

   ```
   svn add [file]
   ```

   If you just add a new file in one directory in the working copy, this file will not be added to it. You have to do this, so subversion knows, it belongs to it. You understand the meaning of this if you work with emacs, that creates backup of files in the same directory.
   Important is, that you have to commit it, so that other people see the change.

6. delete, del, remove, rm
   The oposite of add

   svn delete

   ```
   svn delete [file]
   ```

   Important, if you accidentally delete a file with "rm [file]" you can call it back with "svn update" and then delete it with "svn delete [file]"

7. status, st shows all the files, that have been changed or added, or that are just loosely swimming arround in the working copy.

8. diff
   shows you the differences you have done since the last update.

   svn diff

   ```
   svn diff [file]
   ```

   Leave [file] away to see all changes.
   If you want to see all changes since a certain revision, you can add the parameter -r [revisionnumber] to it.

9. copy, cp
   As the name implies, it copies some part of the working copy somewhere else.

svn cp

```
svn cp [source] [destination]
```

What can come in handy, is that source may also be an entire path. You do not have to include a -R to recursivley add its contents.

10. move, mv
    Same as copy, but moves things around.

11. all other things
    Everything else is too advanced to be mentioned here, and as such will only be for complex svn transactions, between different structure.

    svn cp

    ```
    http://subversion.tigris.org/servlets/ProjectDocumentList
    ```

    This site has a great manual about all those actions. Visit it and read it if you like.

1.3.3 conclusion

Now you have a slight oversight of the most important commands of svn. They pretty much describe, what a beginner should know, before workling with subversion. Over time you know those subcommands by heard, and you will beginn to love them all.
What remains to be said, is that tortoiseSVN for windows does not need all the flags, but there too, you have to know what all the commands are about.
I hope you have some fun exploring all the advantages of subversion, and that you manage to create better and more complex programs with it.

2 Subversion in DataCore-style

First of all I have to point out, that this is not a unique way of doing subversion control, and if you use this chema, you will understand, why it is a fantastic approach to code like this in a team.

2.1 Structure

DataCore splits the programming into 4 pieces:

1. trunk
   In the trunk only the chief programmer has a say. He controls it, and only he can code into it. Out of the trunk there will be created the releases, if the programm is in such an advanced state.

2. branches
   Here all the Developpers create their stuff. Usually, everyone gets his own branch, either one that is named after his name, or one that is named by the Part he is creating
   As an example, when I am joining a project, and i just want to try some things with the code, I call my branch: branches/myname. Over time I advance and understand the code, and I am ready to do my part. I decided to do the part of the programm, that manages the display, so I name the branche: branches/display.
   I think you get the point.

3. tags This is nothing else, than a copy of the trunk. The only difference here, is that this wont evolve over time. You can say, that a tag is a copy of the trunk at a certain revision.
   They are usually called tags/[versionnumber]

4. releases
   This is the same thing as a tag, with the slight difference, that it is a real release. With this you are going public. Usually you don't do this with a tag.

2.2 Example

A little example of this is, the Linux Kernel. (although it is held in CVS)
There are thousands of people working on it in branches, but only one person is capable, of adding new features into it's trunk. Tags are created between releases of new Kernels (example: the mm-sources). And releases are always named with "'2.6.4,2.5.5"' and so on.
So you see, Linus Torvards is the Lord of the trunk, and he is the only one who can make a release. All the others can only work in branches, and can only make tags.
Pretty cool, ey?

2.3 Working with the code at DataCore

The infrastructure of DataCore is given to us, so we have to give to it. And we have to make this in a nice, polite and serious. So here are the things you have to do, and respect:

2.3.1 checking out the code

As I have allready told you, the first thing you do is to check out the Source. You can do this either as an anonymous user in read-only mode via:

svn checkout

```
svn checkout https://open.datacore.ch/read-only/orxonox[1] orxonox
```

Or if you have a registered username and a password:

svn checkout

```
svn checkout https://open.datacore.ch/pw/orxonox orxonox
```

Be sure to accept the certificate, and if you like, to save the authentication information.

2.3.2 Coding

Hold yourself to the coding conventions of the GNU Public License. If you don't know what this means just code in a structured order, and ask someone, what you should look out to.
Always consider, taht others might work with your code, and they have to understand it too.

2.3.3 Checking in

Checking in is quite easy, the only thing, that you have to do is:

svn commit

```
svn commit -message "message"
```

Important here is, that the message is structured in the right way, so one can see at first sight what you commited. A message should look like this

message

```
[Projectname]/[where]: [Topic]
```

An example would be:

examle

```
svn ci -m "orxonox/branches/opengl: Made the reflections look perfect."
```

Hold yourself to this, because the world will know what you are doing...

3 Afterword

I hope this guide will help you in learnign how to handle the principles of Subversion. If not please mail to us at orxonox-dev@mail.datacore.ch. Also all constructive comments, positive or negative are welcome there. I know that my english is not the best by its nature, so I hope you send me all errors, or suggestins about better sentences that you can find.

[1] this implies, that you will check out project orxonox