

*Secure Data Storage and Distribution
Using Revision Control Systems*

Lasse Kliemann
lasse-subversiontalk-2007@plastictree.net

While working on a project, it is nice to...

- ▶ ... be able to switch back to **older versions** of a file, or to compare different versions against each other.
- ▶ ... have an easy procedure for doing **backups**.
- ▶ ... access your files from **anywhere** over the network (reading and writing).
- ▶ ... work with **several persons** on the same set of files, without data loss and with some kind of **conflict management**.

All this and more can be realized with **revision control systems** (e.g., CVS, subversion, GNU arch, darcs) and tools for remote access (e.g., SSH).

This talk focusses on subversion and SSH.

The Naive Way

- ▶ The user opens a new file and starts writing.
- ▶ The more work he spends on the file, the more precious it becomes.

This precious content is available in only **one place**, namely in the file on the disk (and during editing also in the RAM of the machine).

Think of...

- ▶ the user deleting the file accidentally
- ▶ another user deleting the file (accidentally or maliciously)
- ▶ a malicious program (e.g., a virus) deleting the file
- ▶ the harddisk malfunctioning (can happen at any time!)

But I Have a Backup!

Backups are fine. However...

- ▶ How often do you back up your files?
- ▶ How comfortable or flexible is your backup procedure?
- ▶ How well-organized are your backup collections?

Not to mention the mess when working on the same set of files

- ▶ in different locations
- ▶ with several persons

at the same time.

Basic Idea of Revision Control Systems

- ▶ **Repository:**
 - ▶ Contains "everything" about the project.
 - ▶ Is never accessed directly, but only via special tools. Subversion provides the `svn` command for this.
 - ▶ May be on a remote machine. (This is the preferred setup.)
- ▶ **Working Copy:**
 - ▶ Obtained from the repository: `svn co URL`
This is called a **checkout**.
 - ▶ Directory tree stored in home directory.
 - ▶ Accessed normally (with some exceptions).
 - ▶ There can exist several working copies (on different machines) at the same time.
 - ▶ Changes are made known to the repository by a subversion command: `svn commit` or `svn ci`
This is called a **commit**.

Each commit creates a new **revision** of the directory tree (inside of the repository). Once committed, a revision becomes **immutable** inside the repository.¹

Thus, we have:

- ▶ A history:

We can obtain files from older revisions from the repository.

- ▶ A backup:

- ▶ Files are stored in the working copy **and** in the repository.
- ▶ They are immutable inside the repository.
- ▶ If you have a system administrator, he will do backups of the repository.

¹...provided that the repository is only accessible via `svn` (and `svnserve`). Your system administrator should set the repository up in this way. Administrators see here please: <http://svnbook.red-bean.com/nightly/en/svn-book.html#svn.serverconfig.svnserve.sshtricks>

Sample Session

Assume you are going to write an article on the topic *foo*. Ask your system administrator to create a repository for it. He will give you an URL for the new repository, which might look something like this: `svn+ssh://account@svnmachine.heaven.org/foo`

Then you do:

- ▶ Check out a working copy.
`$ mkdir work; cd work`
`$ svn co svn+ssh://account@svnmachine.heaven.org/foo`
- ▶ Edit a file inside of the working copy.
`$ cd foo`
`$ vi article.tex`
- ▶ Tell subversion that the new file shall be part of the project.
`$ svn add article.tex`
- ▶ Do a commit.
`$ svn ci`

- ▶ New files have to be scheduled for a commit using `svn add`.
- ▶ `svn status` shows which files have been modified, added, or are unknown.

```
$ vi article.tex  
$ xfig figure1.fig  
$ svn add figure1.fig  
$ latex article.tex  
$ svn status
```

```
M   article.tex  
?   article.dvi  
A   figure1.fig
```

Concurrency

Assume that several members of your team work on the project files – everyone using his own working copy.

- ▶ A so-called **update** incorporates changes that others committed to the repository into your own working copy:

```
$ svn up
```

- ▶ You let others know about your changes by committing them to the repository.

However: you cannot commit a file that has been modified in the repository (by the commits of other members) since your last update (or checkout) without doing an update first!

Merging and Conflicts (1)

Assume that `article.tex` has local and remote changes, i.e., it has been modified in your working copy and in the repository since your last update (or checkout).

If you are lucky, subversion is able to **merge** the two versions.

- ▶ `$ svn up`

```
G    article.tex
```

- ▶ The file can now be committed (maybe after a quick inspection).

Merging and Conflicts (2)

If the two sets of changes intersect, subversion notes a **conflict**.

▶ `$ svn up`

```
C      article.tex
```

▶ You now have to resolve the conflict by hand (by editing `article.tex`).

▶ When you are certain that everything is OK, do:

```
$ svn resolved article.tex
```

▶ The file can then be committed.

▶ In case of a conflict, usually a log entry should be made on the following commit.

▶ With a bit of coordination among the members of your group, conflicts can be kept rare.

Resolving Conflicts (1)

- ▶ Sally and Peter work on the same file `article.tex`.
- ▶ ...in different working copies of course.
- ▶ Both check out this version:

Let $\epsilon > 0$,
then there exists an algorithm for
the problem of ...

Resolving Conflicts (2)

They work on the file in parallel.

- ▶ Sally corrects one mistake:

```
Let  $\epsilon > 0$ ,  
then there exists an algorithm for  
the problem of ...
```

- ▶ Peter corrects the other one:

```
Let  $\epsilon > 0$ ,  
then there exits an algorithm for  
the problem of ...
```

Resolving Conflicts (3)

- ▶ Sally commits first.
- ▶ Later, Peter tries to commit:

```
$ svn ci
```

```
Sending          article.tex
```

```
svn: Commit failed (details follow):
```

```
svn: Out of date: 'article.tex' in transaction '5-1'
```

Resolving Conflicts (4)

- ▶ Peter makes an update:

```
$ svn up
```

```
C    article.tex
```

```
Updated to revision 5.
```

- ▶ Peter's working copy of `article.tex` now looks like:

```
Let  $\epsilon > 0$ ,  
<<<<<< .mine  
then there exists an algorithm for  
=====  
then there exists an algorithm for  
>>>>>> .r5  
the problem of ...
```

Resolving Conflicts (5)

- ▶ Peter merges the two versions by hand:

```
Let  $\epsilon > 0$ ,  
then there exists an algorithm for  
the problem of ...
```

- ▶ He tells subversion that the conflict is resolved.

```
$ svn resolved article.tex  
Resolved conflicted state of 'article.tex'
```

- ▶ Finally, he is allowed to commit.

```
$ svn -m "merged typo corrections" commit  
Sending article.tex  
Transmitting file data .  
Committed revision 6.
```

Resolving Conflicts (6)

- ▶ Later, Sally makes an update.

```
$ svn up
```

```
U    article.tex
```

```
Updated to revision 6.
```

- ▶ Her working copy now looks just like Peter's.

Let $\epsilon > 0$,
then there exists an algorithm for
the problem of ...

Older Versions

- ▶ Show all changes made to the working copy since the last checkout or update:
`$ svn diff`
- ▶ Show all differences between the working copy and an older revision, e.g., revision 5:
`$ svn -r5 diff`
- ▶ Retrieve complete contents of `article.tex` from revision 5 (printed on the screen or redirected to a file):
`$ svn -r5 cat article.tex`
`$ svn -r5 cat article.tex > article.tex.rev5`
- ▶ You can also check out an entire older revision:
`$ svn -r5 co URL`

Log Entries

- ▶ Subversion requests a log entry from you for every commit.
- ▶ To this end, it starts an editor on every commit.
- ▶ The log entries can be viewed using:
`$ svn log [file]`
- ▶ Can supply a log entry directly using switches:
`svn commit -m msg`
`svn commit -F logfile`
- ▶ If you do not care about log entries, your system administrator can install a shell function or alias in order that a call to `svn ci` will generate empty log entries by default.

More on Working on the Working Copy

- ▶ Instead of `cp`, `mv`, and `rm`, use subversion commands:
\$ `svn cp file1 file2`
\$ `svn mv file1 file2`
\$ `svn rm file`
- ▶ `svn cp` can be used to create a **branch** or to **tag** a revision.
- ▶ Tell subversion to ignore certain files, e.g., `*.aux` files:
\$ `svn propedit svn:ignore .`
- ▶ Working copies contain in each directory a `.svn` subdirectory. These should not be modified directly.

In Case the URL Changes (1)

”Cool URLs don’t change” – however, it may still happen.

Solution 1:

- ▶ Commit all changes before the old repository is closed down.
- ▶ Delete the working copy.
- ▶ Do a new checkout from the new URL.

In Case the URL Changes (2)

Solution 2:

- ▶ Determine the current URL, say, *URL1*:
`svn info`
- ▶ Switch the working copy:
`svn switch --relocate URL1 URL2.`

Overview of Commands

- ▶ `svn co URL`
- ▶ `svn status`
- ▶ `svn up`
- ▶ `svn add file`
- ▶ `svn ci`
- ▶ `svn diff [file]`
- ▶ `svn resolved file`
- ▶ `svn rm file`
- ▶ `svn mv file1 file2`
- ▶ `svn cp file1 file2`
- ▶ `svn mkdir dir`
- ▶ `svn log [file]`
- ▶ `svn info`
- ▶ `svn switch --relocate URL1 URL2`

Traps and Pitfalls (1)

NO: Infrequently updating or committing.

- ▶ Do updates often.
Always update before you begin your work.
- ▶ Commit whenever some part of the work is done.
- ▶ An update means that you get a chance to adapt your work to the changes made by others.
- ▶ A commit means that you give others a chance to do the same regarding their work and your changes.
- ▶ **However:** avoid committing faulty code (under certain conditions)!

Traps and Pitfalls (2)

NO: Large files or long lines.

- ▶ Use multiple smaller files.
- ▶ Include commands (e.g., `\include{}` and `\input{}` in \LaTeX) glue them together.
- ▶ Start a new line often.
- ▶ It's a good idea to start a new line roughly after every clause.
- ▶ This also helps editing a **lot** if you use good line-oriented editors, e.g., the VI-Improved² editor (`vim`).

²<http://www.vim.org>

Traps and Pitfalls (3)

Carefull with binary and/or large files!

- ▶ Many of subversion's features do not work on binary files.
- ▶ Merging does not work, for instance.
- ▶ You can use [locking](#) in case it is necessary to have binary files under subversion's control.

- ▶ Large files make the repository grow – and it is difficult to remove a file from the repository once committed.
- ▶ Think twice before adding and committing large files.

Traps and Pitfalls (4)

NO: Forget to `svn add` a new file.

- ▶ It is OK to have files in the working copy which are unknown to subversion.
- ▶ However, if a new file shall be committed, it must be added:
`svn add file`
- ▶ Beginners often forget this step. Later they wonder why their co-authors cannot see the file.
- ▶ A missing file might temporarily break things.
(Consider you added `\include{foo}` to an existing file, but `foo.tex` cannot be found.)

Traps and Pitfalls (5)

NO: File open in editor during update.

- ▶ Copy of the file is in editor.
- ▶ An update is performed.
- ▶ This alters the file on the disk.
- ▶ However, the copy in the editor stays the same.
- ▶ The file is saved from the editor to disk.
- ▶ The file on disk now is the former one – the changes due to the update are knocked over by this.
- ▶ For subversion, it looks like normal changes to the updated file were made.
- ▶ Later, co-authors wonder where their changes have gone.

Public Key Authentication

- ▶ Create a public/private keypair:³
\$ ssh-keygen -t rsa -b 4096
And go with the default for all questions.
- ▶ Send the public key to the administrator. It is in your home directory in the file: `.ssh/id_rsa.pub`
You can send it by e-mail:
mail admin@heaven.org < .ssh/id_rsa.pub
- ▶ Do not give your secret key to others! (The secret key is in your home directory in the file: `.ssh/id_rsa`)

³Using an RSA key of that length is just a suggestion. Maybe a shorter key suffices. Maybe DSA keys are better. Sound feedback on this is welcome.

Where to Get Help?

- ▶ `http://subversion.tigris.org`
FAQs and mailing lists (with searchable archives).
- ▶ `http://svnbook.org`
- ▶ `$ svn help`
`$ svn help commit`
`$ svn help up`
...