

L_AT_EX and Subversion*

Mark Eli Kalderon

Email eli@markelikalderon.com
Website <http://markelikalderon.com>
Address Department of Philosophy
University College London
Gower Street
London, WC1E 6BT
United Kingdom

Abstract Subversion is a popular open source version control system. When writing complex L_AT_EX documents, it is useful to keep track of their development with a version control system such as Subversion. This article covers installation of Subversion, the creation of a local Subversion repository and working copy, the Subversion workflow, and how to get L_AT_EX to interact with Subversion with the `svn-multi` package.

1 Introduction

Many of the tools that programmers use are, in fact, readily adaptable to the task of writing. Programmers and writers face at least one common problem, they need to track small changes over time. Programmers, being programmers, have written software to meet this particular need, software that can meet the writer's corresponding need. So-called *version control systems* (sometimes also called *source code management*), can help you keep track of the development of complex L_AT_EX documents. Further you can use it as collaborative tool.¹ Once you incorporate version control into your workflow, you will soon find it indispensable.

You might wonder what the big deal is. After all, many save their latest drafts in an *ad hoc* fashion, say, with multiple files. The problem is that this procedure is *ad hoc* and *unreliable*. Version control is not magic, it requires discipline, but it

*Thanks to Francisco Reinaldo for his help and encouragement.

1. See [2] for discussion of collaborative use of Subversion.

provides the framework for keeping track of revisions in a non-*ad-hoc* manner. It may well save you a lot of grief.

*Subversion*² is a popular open source version control system. In this article, I explain how to use Subversion in the production of L^AT_EX documents. In [section 2](#), I describe the basic concepts of Subversion. In [section 3](#), I explain how to install Subversion. In [section 4](#), I explain how to create a local Subversion repository. In [section 5](#), I explain how to check out a working copy. In [section 6](#), I describe the basic Subversion workflow. Finally, in [section 7](#), I explain how the `svn-multi` package³ can help L^AT_EX interact with Subversion.

2 Subversion, Basic Concepts

In Subversion, a directory and its contents are kept in a Subversion *repository*. At the heart of a Subversion repository is a database that tracks not only the directory and its contents but, importantly, changes to that directory. The Subversion repository can reside locally on your machine, or, ideally, on a server.

Two observations are relevant.

First, when a change is committed to the repository, Subversion records four important pieces of information. Remember being taught journalism in high school? A good story must provide answers to four questions: “Who?”, “What?”, “When?”, and “Where?” Or so we were taught. When you commit a change, Subversion records the “Who?”, “What?”, “When?”, and “Where?” of the committed change:

1. Subversion records WHO committed the change
2. Subversion records WHAT the change was (a description of the change)
3. Subversion records WHEN the change was committed
4. Subversion records WHERE the change was made (which files or subdirectories were changed)

With each change, Subversion increments the revision number of the repository. The revision number represents the number of changes that have been made, or committed, to the Subversion repository.

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

3. `CTAN:/macros/latex/contrib/svn-multi`

Second, Subversion tracks changes to a *directory* and its contents. When a change is committed, the revision number of the *entire* repository is incremented. Whereas CVS⁴, Subversion's predecessor, primarily tracks the history of revisions to individual files, Subversion primarily tracks the history of revisions to a directory.

Subversion uses a centralised model of version control. There is a central repository from which a local *working copy* may be *checked out*. You can think of a working copy as your own personal sandbox where you can freely modify the directory and its contents. Once you are satisfied with the changes you have made, you then *commit* these changes to the repository.

Some liken Subversion to a time machine. Wish you hadn't deleted that paragraph? With Subversion, the missing paragraph is just a `svn cat` away.

3 Installing Subversion

Subversion can be built from source or installed with a binary. Binaries exist for a number of operating systems including Red Hat Linux, Fedora Core, Debian GNU/Linux, FreeBSD, OpenBSD, NetBSD, Solaris, IBM i5/OS, Mac OS X, and Windows. A listing of these are available at the Subversion website⁵. One popular binary for Mac OS X is not listed on that page, Martin Ott's binary package⁶.

To build Subversion from source, download the most recent distribution tarball from the Subversion website⁷. Unpack it, and use the standard *nix procedure to compile:

```
$ ./configure
$ make
$ sudo make install
```

Be sure to read the document "Install" first, though, to check for dependencies.

4. <http://www.nongnu.org/cvs>

5. http://subversion.tigris.org/project_packages.html

6. <http://www.codingmonkeys.de/mbo>

7. <http://subversion.tigris.org/servlets/ProjectDocumentList>

4 Creating a Local Subversion Repository

Creating a local Subversion repository is easy. It can be done with the following command:

```
$ svnadmin create /path/to/your/repository/
```

The command `svnadmin create` creates a new empty repository at the path provided as an argument. If the path provided does not exist, then Subversion will create it for you. So be careful: `/path/to/your/repository/` is just a dummy path, to be replaced with whatever you like.

The repository we just created is empty. We can *import* a directory and its contents into the empty repository as follows. First either copy a directory or create a directory in `/tmp/`:

```
/tmp/project/
```

(Again, `project/` is a dummy label.) Once you have populated `/tmp/project/` with whatever content you want, we are ready to begin. To import `/tmp/project/` into the newly created empty repository, we use the following command:

```
$ svn import -m "initial import" /tmp/project/  
file:///path/to/your/repository/
```

The `svn import` command takes two arguments, the path of the directory to be imported and the URL of the repository. It can also take options. The option we have used, `-m`, is used to give a description of the change to the repository. The description follows the `-m` in double quotes. Since the change is an initial import of the directory, that is the description we have used. You can now delete `/tmp/project/` with:

```
$ rm -r /tmp/project/
```

but just to be safe, let's first *check out* the working copy.

5 Checking Out a Working Copy

Checking out a working copy is easy. And it only needs to be done *once*.

First, navigate to the directory where you want your working copy to be. Suppose you want to keep your working copy in your home directory, then use the command:

```
$ cd ~
```

If you want the working copy to reside elsewhere, simply replace the tilde with the desired path.

To check out a working copy, we now use the following command:

```
$ svn checkout file:///path/to/your/repository/ project
```

The command `svn checkout` has two arguments, the URL of the repository and the name of the working copy to be created by Subversion. Your working copy now resides at:

```
~/project/
```

Check the contents of your working copy with:

```
$ ls ~/project/
```

If the contents of your working copy are as expected, you can now safely delete `/tmp/project/`.

6 The Subversion Workflow

Okay. That was a bit of work setting things up. But now the fun begins.

There are four basic components to the Subversion workflow:

1. Updating your working copy
2. Making changes to your working copy
3. Reviewing your changes
4. Committing your changes to the Subversion repository

Let's review these in turn.

6.1 Updating

You should always begin your workflow by updating your working copy. This propagates any changes in the Subversion repository to your working copy. Updating is easy. At the root of your working copy simply issue the command:

```
$ svn up
```

Suppose that `foo.tex` has been changed since your last update, and it is the seventeenth revision of the repository, then the command will yield:

```
U foo.tex
Updated to revision 17
```

Thus, your working copy has been updated to reflect the changes to `foo.tex` and that it now reflects revision 17 of the repository. The letter code U is one of several codes that can be issued. These include:

- U bar — file bar has been updated;
- A bar — file bar has been added;
- D bar — file bar has been deleted;
- G bar — file bar has been updated but you have made local changes in your working copy not reflected in the repository but these changes do not conflict.

6.2 Changing

There are two kinds of changes that can be made in the working copy:

1. Changes to individual files;
2. Changes to the structure of the directory.

Changes to individual files are easy. Just make the changes in the appropriate editing program. You don't need to issue any Subversion specific commands.

Changes to the structure of the directory, however, require Subversion's help (so that Subversion knows what's happening to the directory). Fortunately, the relevant commands are easy to understand. To schedule the file or directory `foo` for addition or deletion from the repository the next time you commit, you use the following commands, respectively:

```
$ svn add foo
$ svn delete foo
```

To create a copy of `foo` called `bar` and schedule the addition of `bar` to the repository the next time you commit, you use the following command:

```
$ svn copy foo bar
```

To move `foo` to `bar`, you use the following command:

```
$ svn move foo bar
```

This creates a copy of `foo` called `bar` and schedules `bar` for addition and schedules `foo` for deletion the next time you commit it.

6.3 Reviewing

Before committing your changes, it is a good idea to review the changes you made. Helpfully, Subversion has provided some useful tools for doing this.

First up is the command:

```
$ svn status
```

If this command is issued at the root of your working copy it will yield a description of all the changes that you have made since last updating. The command yields a list of files and directories preceded by letter codes indicating the nature of the changes made. These include:

- M `foo` — `foo` has been modified since last updating;
- A `foo` — `foo` is scheduled for addition;
- D `foo` — `foo` is scheduled for deletion;
- C `foo` — `foo` conflicts with an update.

If `svn status` indicates that `foo` has been modified, you may want to examine these changes in detail in order to write a helpful commit message.

The command:

```
$ svn diff
```

run at the root of the working copy will print out a list of changes to modified files in `diff` format. Unfortunately, `diff` format displays *line* differences, not *word* differences. This is not great for prose, since paragraphs in text files are long lines so multiple differences in a paragraph will not be adequately represented. Fortunately, Subversion allows you to use external `diff` programs. To use an external `diff` command, preferably one that displays word differences, simply use the option:

```
--diff-cmd CMD
```

where `CMD` is the name of the external `diff` command. If the external `diff` command takes different arguments than `svn diff` you may need to write a wrapper script to use this option.

The command, `wdiff`,⁸ is a frontend to GNU `diff` that displays word differences. It is also possible (and desirable) to use a GUI `diff` program. For example, on Mac OS X, if you install the developer tools, one gem that you will get is FileMerge, a descendant of NEXTSTEP's merge utility. FileMerge provides a visual comparison of text files. FileMerge can be invoked from the command line with `opendiff`. FileMerge, like GNU `diff`, captures line differences, but, like `wdiff`, it also highlights word differences. Unfortunately, `opendiff` and `svn diff` use different arguments. So to get `opendiff` to work with Subversion requires a wrapper script. Fortunately, Bruno De Fraine has done the heavy lifting. His script `fmdiff` is just such a wrapper. `fmdiff` can be downloaded (or checked out) from his website⁹. If you install `fmdiff` in `/usr/local/bin/` or `~/bin/` as is your preference and make it executable, you could then run the command:

```
$ svn diff --diff-cmd fmdiff foo.tex
```

to reveal in FileMerge the changes you have made to `foo.tex` since last committing, as can be seen in [Figure 1](#).

6.4 Committing

So far, you have updated your working copy, made appropriate changes, and have reviewed these changes. If you are happy with these changes, it is time to com-

8. <http://wdiff.progiciels-bpi.ca>

9. <http://ssel.vub.ac.be/svn-gen/bdefrain/fmscripts>

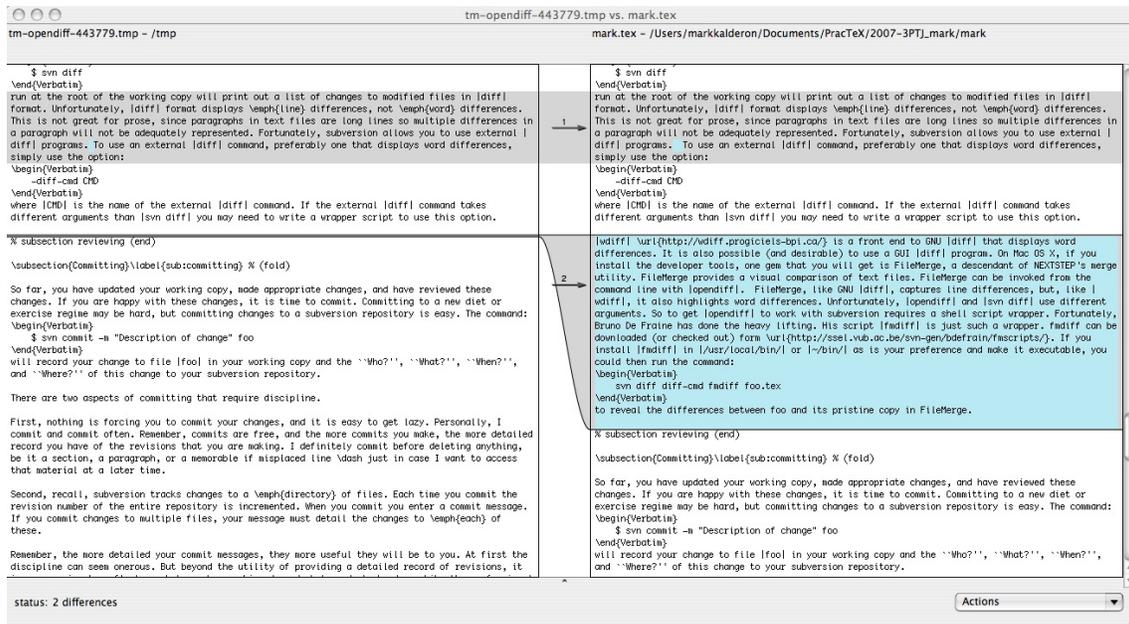


Figure 1: FileMerge.

mit. Committing to a new diet or exercise regime may be hard, but committing changes to a Subversion repository is easy. The command:

```
$ svn commit -m "Description of change" foo
```

will record your change to file `foo` in your working copy and the “Who?”, “What?”, “When?”, and “Where?” of this change to your Subversion repository.

There are two aspects of committing that require discipline.

First, nothing is forcing you to commit your changes, and it is easy to get lazy. Personally, I commit and commit often. Remember, commits are free, and the more commits you make, the more detailed record you have of the revisions that you are making. I definitely commit before deleting anything, be it a section, a paragraph, or a memorable if misplaced line—just in case I want to access that material at a later time.

Second, recall, Subversion tracks changes to a *directory* of files. Each time you commit the revision number of the entire repository is incremented. When you commit, you enter a commit message. If you commit changes to multiple files,

your message must detail the changes to *each* of these.

Remember, the more detailed your commit messages, the more useful they will be to you. At first the discipline can seem onerous. But beyond the utility of providing a detailed record of revisions, it is an occasion to reflect on what you have achieved, and what needs to be done. Like the confessional for Catholics, or the weekly review for GTD geeks, the exercise of formal reflection can be good for you.

6.5 Summary

The Subversion workflow involves four basic components: updating, changing, reviewing, and committing. We have seen how to manage these on the command line. While the GUI's supplanting the command line represents the triumph of the Image over the Word, the GUI has its place—even in text editing. With respect to Subversion, there's cognitive utility in being able to visualise the structure of your repository or working copy. (For an extensive listing of GUI front ends on a variety of platforms see the Subversion website¹⁰.) While there are GUI front ends for Subversion, none are perfect. A compelling solution to graphically representing Subversion repositories and working copies has yet to be found. And no GUI front end has the power, speed, and flexibility of Subversion on the command line. So take the time to learn the Subversion commands. The basics can be picked up in an afternoon.

7 The svn-multi Package

L^AT_EX is great for managing complex documents. When working on a complex L^AT_EX document, it is important, nay, *imperative* to keep it in some form of version control. (The necessity may not be apparent until you actually *use* a version control system such as Subversion—if you don't, well, just trust me, at least for now.)

If you are keeping your complex L^AT_EX document in a Subversion repository, it would be useful to include information about the current revision, the "Who?", "What?", "When?", and "Where?" of the last committed change, within your

10. <http://subversion.tigris.org/links.html>

document. You could enter this information by hand, but that would be tedious and unreliable. Fortunately, there are L^AT_EX packages that can help. There are three:

1. `svn`¹¹
2. `svninfo`¹²
3. `svn-multi` (aka `svnk`)¹³

The `svn-multi` package suits my workflow best. Here's why. The `svn` package doesn't work if your L^AT_EX document comprises multiple files. Since I sometimes work with a master document that includes several separate files, that's out. The `svninfo` and `svn-multi` packages do not share this limitation. The `svninfo` package, however, itself has an important limitation. It only registers revision information when you *check out* the document. Thus, if you create a L^AT_EX document, add it to your working copy, and commit the addition, the revision information, the "Who?", "What?", "When?", and "Where?", will never be registered. So that's out. That leaves `svn-multi`—which has worked well for me. Let me detail how I use it.

The `svn-multi` package does two things:

1. It registers revision information, the "Who?", "What?", "When?", and "Where?";
2. It displays this information in your L^AT_EX document.

7.1 Registering Revision Information

In the preamble, I have the following:

```
\usepackage{svn-multi}
\svnidlong
{$LastChangedBy$}
{$LastChangedRevision$}
{$LastChangedDate$}
{$HeadURL$}
```

11. CTAN:<http://tug.ctan.org/tex-archive/macros/latex/contrib/svn>

12. CTAN:<http://tug.ctan.org/tex-archive/macros/latex/contrib/svninfo>

13. CTAN:<http://tug.ctan.org/tex-archive/macros/latex/contrib/svn-multi>

The command:

```
\svnidlong
{$LastChangedBy$}
{$LastChangedRevision$}
{$LastChangedDate$}
{$HeadURL$}
```

registers WHO made the change, WHAT the change was, WHEN the change was made, and WHERE the change is—the URL of the changed file. (Strictly speaking, `{$LastChangedRevision$}` registers, not what the change was, but a revision number that represents this. If `n` is the revision number, a detailed description of what has changed will be given by the command `svn log -r n foo.tex`, where `foo.tex` is the name of your L^AT_EX source)

Now we need to get these commands to interact with Subversion. To do so, navigate to the directory containing your L^AT_EX source, `foo.tex` and enter the following command in the terminal *once*:

```
$ svn propset svn:keywords 'LastChangedBy LastChangedRevision
LastChangedDate HeadURL' foo.tex
```

This will return the following:

```
property 'svn:keywords' set on 'foo.tex'
```

When you next commit a change to your repository or receive a change by updating your working copy, this information will be registered.

7.2 Displaying Revision Information

Now that we have registered the revision information, we want to use this information, to display it somewhere appropriate in our L^AT_EX document. Fortunately, the `svn-multi` package provides commands to do just that.

We want to display the following information:

- the author of the revision;
- the revision number of the last change;
- the date of the last change;

– the URL of the file.

The command:

```
\svnauthor
```

displays the author registered by:

```
{$LastChangedBy$}
```

in the preamble. This is the author of the last committed change to the document.

The command:

```
\svnrev
```

displays the revision number registered by:

```
{$LastChangedRevision$}
```

in the preamble. This is the revision number of the last committed change to the document. The command:

```
\svndate
```

displays the date registered by:

```
{$LastChangedDate$}
```

in the preamble. This is the date of the last committed revision in the repository. Finally, the command:

```
\svnkw{HeadURL}
```

displays the URL registered by:

```
{$HeadURL$}
```

in the preamble. This is the URL of the file as it resides in the Subversion repository.

Let's consider one possible use of these commands and some possible refinements. Consider the following minimal example:

```

\documentclass{article}
\usepackage{svn-multi}
\svnidlong
{$HeadURL$}
{$LastChangedDate$}
{$LastChangedRevision$}
{$LastChangedBy$}
\begin{document}
    Hello World!
\end{document}

```

Suppose we want to include the revision information in a final footnote of the article. We might append the following command to Hello World!:

```
\footnote{\svnauthor \svnrev \svndate \svnk{HeadURL}}
```

That's okay, to a first approximation. But it would be useful to label this information. Thus, for example, the value of:

```
\svnrev
```

is just a number. A label would provide some useful context for understanding what that number represents. Indeed, labels would provide useful context for understanding each piece of information. Moreover, it would be useful to clearly distinguish this information, so let's separate them with semicolons. This gives us:

```
\footnote{Author: \svnauthor; Revision: \svnrev; Last changed
on: \svndate; URL: \svnk{HeadURL}}
```

Two further refinements. If we load the hyperref package and use the `\url` command we can add a hyperlink to the file in the Subversion repository:

```
\footnote{Author: \svnauthor; Revision: \svnrev; Last changed
on: \svndate; URL: \url{\svnk{HeadURL}}}
```

And if we load the url package, urls will be correctly formatted.

Our minimal example would then be as follows:

```

\documentclass{article}
\usepackage{svn-multi}
\svnidlong
{$HeadURL$}
{$LastChangedDate$}
{$LastChangedRevision$}
{$LastChangedBy$}
\usepackage{url}
\usepackage{hyperref}
\begin{document}
  Hello World!\footnote{Author: \svnauthor; Revision:
  \svnrev; Last changed on: \svndate; URL:
  \url{\svnkw{HeadURL}}}
\end{document}

```

Earlier, I mentioned that one advantage that the `svn-multi` package has over the `svn` package is its support for L^AT_EX documents that comprise multiple files. So, suppose you have a master file, `master.tex`, that *includes* the files, `chapter1.tex` and `chapter2.tex`. All three of these files need to be in version control, and so we need to register the revision information for *each*. Thus the command:

```

\svnidlong
{$HeadURL$}
{$LastChangedDate$}
{$LastChangedRevision$}
{$LastChangedBy$}

```

needs to occur in `master.tex`, `chapter1.tex`, and `chapter2.tex`. And for each of these documents you need to set the `svn` keywords. So you would need to run:

```

$ svn propset svn:keywords 'LastChangedBy LastChangedRevision
LastChangedDate HeadURL' master.tex
$ svn propset svn:keywords 'LastChangedBy LastChangedRevision
LastChangedDate HeadURL' chapter1.tex
$ svn propset svn:keywords 'LastChangedBy LastChangedRevision
LastChangedDate HeadURL' chapter2.tex

```

Please see the documents, `example_main.tex`¹⁴, `example_chap1.tex`¹⁵, and `example.pdf`¹⁶ that come with the `svn-multi` package for a detailed example of usage.

7.3 Initial Workflow

Call our minimal example `example.tex`. To get the desired result, the following steps must be taken:

1. Add `example.tex` to your Subversion repository:

```
$ svn add example.tex
```

This should return:

```
A example.tex
```

2. Commit the addition:

```
$ svn commit -m "Added example.tex" example.tex
```

This should return:

```
Adding example.tex
Transmitting file data.
Committed revision n
```

where `n` is a dummy letter representing the current revision number of your repository.

3. Set the Subversion keywords:

```
$ svn propset svn:keywords 'LastChangedBy LastChangedRevision
LastChangedDate HeadURL' example.tex
```

This should return:

```
property 'svn:keywords' set on 'example.tex'
```

4. Update the working copy:

14. CTAN:http://www.cam.ctan.org/tex-archive/macros/latex/contrib/svn-multi/example_main.tex

15. CTAN:http://www.cam.ctan.org/tex-archive/macros/latex/contrib/svn-multi/example_chap1.tex

16. CTAN:<http://www.cam.ctan.org/tex-archive/macros/latex/contrib/svn-multi/example.pdf>

```
$ svn up
```

This should return:

```
At revision n
```

5. Check the status of the working copy:

```
$ svn status
```

This should return:

```
MM example.tex
```

meaning that `example.tex` has been modified (since the values of the Subversion keywords have been added to the variables in the `\svnidl` command).

6. Commit these changes:

```
$ svn commit -m "Svn keywords set on example.tex" example.tex
```

This should return:

```
Sending example.tex  
Transmitting file data.  
Committed revision n+1
```

7. Typeset `example.tex` *twice*. The first run generates an auxiliary file called `example.svn` whose content is added to the output of `example.tex` on the second run.

8 Conclusion

Some form of version control should really be at the heart of any writer's workflow. This article has discussed some of the basics of using one popular version control system, Subversion, with \LaTeX . I have really just scratched the surface. There is a lot more useful functionality. For more information about Subversion see Collins-Sussman, et al. [1]. This book is available free online in both HTML and PDF format¹⁷. The package documentation for `svn-multi` is available

17. <http://svnbook.red-bean.com>

at CTAN¹⁸. And remember, commits are free — commit often.

References

- [1] Ben Collins-Sussman, Brian W. Fitzpatrick and C. Michael Pilato, 2004. *Version Control with Subversion*. Sebastopol, CA: O'Reilly. URL: <http://svnbook.red-bean.com/>
- [2] Charilaos Skiadas, Thomas Kjosmoen, and Mark Eli Kalderon, 2007. "Subversion and TextMate: Making collaboration easier for L^AT_EX users". *The PracT_EX Journal*, 3.

18. CTAN:<http://www.ctan.org/tex-archive/macros/latex/contrib/svn-multi/svn-multi.pdf>