The `vc` bundle*

Roland Hieber† Stephan Hennig‡

June 11, 2018

Abstract

This is a script based approach to version control for TEX documents. It works more reliably than keyword substitution based approaches, since it tracks *all* files in a working copy, not only `.tex` files. The `vc` bundle works with LATEX and plain TEX. Currently, Bazaar, Git, Mercurial and Subversion are supported.

Contents

1 Introduction

There is an inherent problem with LATEX and version control software as soon as you're dealing with files generated by an external tool, *e.g.*, graphics. Packages such as `svn-multi` can't track neither binary files, nor source files of graphics compiled by, *e.g.*, MetaPost. For that reason, if you check-in a new revision that only touches a graphic file, your VCS package would never know a check-in has happened and tell you the old revision number (or date or other meta data) in your documents.

---

[*] This document describes the `vc` bundle v0.6

[†] rohieb+ctan@rohieb.name

[‡] stephanhennig@arcor.de

At least, the problem is only of temporary nature and as soon as you check-in a file that is tracked by the VCS package, *i.e.*, a `.tex` file, you'll get the correct revision number again. But traditional VCS packages, that build on the keyword substitution feature provided by some VCS can't track revision information *reliably*, as they look at `.tex` files only.

To enable reliable tracking of revision information one has to look at *all* files in a working copy. Since for non-source files the keyword substitution feature doesn't work, another approach has been taken here. This bundle consists of some scripts that directly talk with the VCS backend to get the desired information and write them to a file `vc.tex`. This file can then be included into your document sources.

The `vc` bundle works with LaTeX as well as plain TeX. Currently, Bazaar, Git, Mercurial and Subversion are supported. Additional contributions are welcome!

2 Usage

Doing version control with the `vc` bundle is very easy. While for other LaTeX VCS packages you need to activate keyword substitution and modify all `.tex` source files, these steps aren't necessary for the `vc` bundle. You just have to copy two files to your project's working copy and add one line to your TeX preamble.

2.1 Installation

Ok, lets set-up the project repository. As a prerequisite, the scripts of the `vc` bundle need `GNU awk`. Please install this first.[1]

The `vc` bundle consists of three files: a shell script, an AWK script, and an automatically generated TeX file. For Unix and Windows the set of files might be

Unix	Windows
`vc`	`vc.bat`
`vc-bzr.awk`	`vc-bzr.awk`
`vc.tex`	`vc.tex`

---

[1] For Windows you can find `gawk` in the `GNUWin32` utilities. Alternative ports can be found in `Msys` or `Cygwin`.

Note, depending on your VCS the AWK script might be any of `vc-bzr.awk`, `vc-git.awk`, `vc-hg.awk` or `vc-svn.awk`. Additionally, while the AWK scripts have the same names on Unix and Windows, the `vc` bundle provides them with different line endings. So, watch out to take the right one for your VCS and OS.

Installation is a one-step procedure (with two additional optional steps).

1. Copy the two script files
    - `vc` (or `vc.bat` for Windows)
    - `vc-bzr.awk` (or `vc-git.awk` or `vc-hg.awk` or `vc-svn.awk`)

   into the top-level directory of your project's working copy.
2. *(Optionally)* You can instruct your VCS software to ignore all three `vc` related files. Please consult the manual of your VCS software about this. Question 10 in Section C contains some brief instructions for Bazaar, Git, Mercurial and Subversion, too.
3. *(Optionally)* Personally, the author is used to check-in both scripts into each project repository to have them available when they are needed and ignore file `vc.tex` only.

2.2 Preparing documents

What remains to be done is adding this line

```
\input{vc}
```

to your main LaTeX or plain TeX document. That's it.

Congratulations! You have now access to several macros containing VCS information in your TeX document. The general macros available are shown in table 1.

The most prominent information is probably the revision number, that can be found in macro `\VCRevision`. For Bazaar and Subversion this is a plain number, for Git it is a 7-hexdigit hash (the truncated 40-hexdigit SHA1 commit hash), for Mercurial it is a 12-hexdigit hash (truncated from 40 symbols). Another macro that might be of interest is `\VCRevisionMod`. This macro is discussed in detail in appendix B.

The remaining macros found in table 1 contain author, date (in different formats) and time of the last commit and should be straightforward to use.

The above mentioned macros are available for all supported systems and in general should be sufficient. However, depending on the VCS software

macro	meaning
`\VCRevision`	current (maximum) working copy revision number
`\VCAuthor`	author of the last check-in operation
`\VCDateRAW`	date of last check-in in native format of the VCS software
`\VCDateISO`	date of last check-in in ISO format YYYY-MM-DD
`\VCDateTEX`	date of last check-in in TeX format YYYY/MM/DD
`\VCTime`	time of last check-in
`\VCRevisionMod`	as `\VCRevision`, but with an additional note if the working copy contains modified files
`\VCModifiedText`	contains the note shown in macro `\VCRevisionMod` if there were modified files. This macro can be redefined by the user.
`\VCModified`	`0` if there are no modified files in the working copy directory; `1` or `2` if there are modified files. In general you don't need this macro.

Table 1: General version control macros.

you are using, there might be additional meta data available.[2] Those data are stored in other macros that are discussed in appendix A.

2.3 Compiling documents

Before file `vc.tex` can be loaded in the document preamble, it needs to be generated. Doing that is as easy as running the shell script `vc` – or `vc.bat` for Windows – before (La)TeX. There are three ways to do this:

1. from a `Makefile` – this is the preferred method,
2. via `\write18` – another automatic solution,
3. manually – not recommended (inconvenient and error-prone).

Here's how the script can be called from within a LaTeX run via the `\write18` feature. Add the two lines

Unix

```
\immediate\write18{sh ./vc}
\input{vc}
```

Windows

```
\immediate\write18{vc.bat}
\input{vc}
```

---

[2] Such as the complete 40-hexdigit SHA1 commit hash for Git or Mercurial.

to your document. If LaTeX sees the first line, it immediately executes the argument of `\write18` on the command-line. That is, the script `vc` – or `vc.bat` – is executed and file `vc.tex` is updated. On the second line LaTeX reads-in the newly generated file `vc.tex`.

To make this work the `\write18` feature has to be enabled. By default, it is disabled for security reasons. For MiKTeX `\write18` can be enabled by calling LaTeX via

```
> latex -enable-write18 ⟨document⟩
```

For other LaTeX distributions, please consult the documentation.

The `\write18` feature is not relevant if `vc` is called by a `Makefile`.

Happy TEXing!
Stephan Hennig

A Notes on supported VCS

To be completed. VCS specific macros are prefixed `\BZR`, `\GIT`, `\HG` or `\SVN`. Tables 2 to 5 show the additional macros available, depending on your VCS.

The macros marked by an asterisk might contains sensitive information such as the path to a repository, file names, *etc.* These macros are only written to file `vc.tex` in *full mode*. By default, full mode is disabled. That is, distributing file `vc.tex` along with your TEX source files should be fairly save, by default.

To activate full mode script `vc` has to be called with command-line option `-f`. This option should only be used with care.

macro	meaning
*`\BZRBranchNick`	branch nickname
`\BZRRevisionId`	full revision id
`\BZRDate`	date of the last revision
`\BZRBuildDate`	current date
`\BZRRevNo`	revision number

Table 2: Bazaar specific version control macros.

macro	meaning
`\GITHash`	40-hexdigit SHA1 commit hash
`\GITAbrHash`	abbreviated commit hash
`\GITParentHashes`	parent hashes
`\GITAbrParentHashes`	abbreviated parent hashes
`\GITAuthorName`	author name
`\GITAuthorEmail`	author e-mail
`\GITAuthorDate`	author date
`\GITCommitterName`	committer name
`\GITCommitterEmail`	committer e-mail
`\GITCommitterDate`	committer date

Table 3: Git specific version control macros.

macro	meaning
`\HGHash`	40-hexdigit SHA1 commit hash
`\HGAbrHash`	abbreviated commit hash
`\HGBranch`	commit branch
`\HGFirstParentHash`	first parent hash
`\HGSecondParentHash`	second parent hash (*All zeroes means one-parent revision*)
`\HGAbrFirstParentHash`	abbreviated first parent hash
`\HGAbrSecondParentHash`	abbreviated second parent hash
`\HGAuthorName`	author name
`\HGAuthorEmail`	author e-mail
`\HGAuthorDate`	author date

Table 4: Mercurial specific version control macros.

macro	meaning
*`\SVNPath`	path to an arbitrary file or directory, that is part of the last commit
*`\SVNName`	`\SVNPath`'s name without path
*`\SVNUrl`	path of `\SVNPath` in the repository
*`\SVNNodeKind`	node kind of `\SVNPath` (file, directory, *etc.*)
*`\SVNRepositoryRoot`	repository root URL
`\SVNRevision`	revision number of `\SVNPath`
`\SVNLastChangedRev`	revision number of `\SVNPath`
`\SVNLastChangedAuthor`	author of the last commit
`\SVNLastChangedDate`	date of the last commit
`\SVNRepositoryUuid`	repository UUID

Table 5: Subversion specific version control macros.

B Checking for local modifications

Some people prefer to be notified, if a document is compiled from a dirty working copy, *i.e.*, from a state not corresponding to a committed revision. This feature has been implemented in the `vc` bundle, but is disabled by default (see below).

Macro `\VCRevisionMod` is similar to `\VCRevision`, but it has an additional message appended to the revision number, if there are any modified files in the working copy.

The actual message is defined in macro `\VCModifiedText` and can be redefined by the user. The default definition is

```
\gdef\VCModifiedText{\textcolor{red}{with local modifications!}}
```

That is, package `color` has to be loaded in the document preamble if macro `\VCRevisionMod` is used or macro `\VCModifiedText` has to be redefined accordingly.

By default, searching for local modifications is disabled to prevent slowing down execution of the scripts. To check a working copy for modified files script `vc` has to be called with the switch `-m`. The `\write18` example from section 2.3 now reads:

Unix

```
\immediate\write18{sh ./vc -m}
\input{vc}
```

Windows

```
\immediate\write18{vc.bat -m}
\input{vc}
```

Note, since distributing documents not corresponding to a committed revision is bad style, it is wise make sure by other means (a release procedure), that distributed documents never contain uncommitted changes. Therefore, a note, say, next to the revision number, doesn't really provide any additional information. If you think you need such a note, something might be wrong with your release procedure.

C Questions and answers

1. **How often do I need to run the script `vc`? Every time before TEX is run?**

2. **Why are VCS data not updated in my document?**
3. **How can I print VCS data at arbitrary places in my document?**
4. **I want to have VCS data in the document only in draft mode!**
5. **How can I change the date format?**
6. **How can I access software specific VCS information, *e.g.*, Git's 40-hexdigit commit hash?**
7. **Why are macros defined with `\gdef` instead of `\newcommand` in file `vc.tex`?**
8. **Macro `\VCRevisionMod` only shows the revision number, even if there are modified files in my working copy.**
9. **In macro `\VCRevisionMod`, how can I get rid of the horizontal skip between revision number and the message?**
10. **How do I ignore files with my VCS?**
11. **Is it possible to get per-file revision data with the `vc` bundle? This were quite handy when working with a multi-file document (say, each file is a chapter). After changing one file, the information could be used to check if an old print-out of another chapter is current.**
12. **Can I put files of the `vc` bundle into a private, public or commercial repository?**

1. How often do I need to run the script `vc`? Every time before TEX is run?

First, it is not recommended to run the script manually, but automatically, either from a `Makefile` or directly from LATEX via `\write18`. For that reason, it shouldn't matter how often the script is called.

To answer the question: If you run the script `vc` manually, it is sufficient to do that once after each check-in or update operation. The only advantage of running the script before *every* TEX run is that it keeps macro `\VCRevisionMod` up-to-date w.r.t. local modifications (when called with the `-m` switch).

Therefore, a fourth way to run script `vc` were to put it into a VCS hook that is called after check-in or update operations. There are no examples for this solution, since it heavily depends on the underlying VCS. Additionally, some VCS might not provide enough hooks to cover all operations that modify a working copy.

2. Why are VCS data not updated in my document?
Make sure script `vc` is run between check-in operations and the TeX run. In case the script is called from TeX via `\write18` (see section 2.3), you've probably just forgotten to enable that feature. Please, refer to the manual of your TeX distribution to learn how to enable `\write18`.

3. How can I print VCS data at arbitrary places in my document?
This question is covered in the UK-TeX-FAQ. Depending on where you want to put VCS information—header, footer, page background—you might be interested in the following links:

- `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=fancyhdr`
- `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=watermark`
- `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=abspos`

Here is some code to put the revision number together with check-in date and time into the foot line with packages `fancyhdr` and `scrpage2`:

```
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyfoot[LE,LO]{Rev: \VCRevision}
\fancyfoot[RE,RO]{Time: \VCDateISO \VCTime}
```

```
\usepackage{scrpage2}
\pagestyle{scrheadings}
\lefoot{Rev: \VCRevision}
\lofoot{Rev: \VCRevision}
\refoot{Time: \VCDateISO \VCTime}
\rofoot{Time: \VCDateISO \VCTime}
```

Another source of information might be the TeX catalogue:

- `http://texcatalogue.sarovar.org/bytopic.html#revision`

In this manual, the `prelim2e.sty` package has been used to present VCS information. See file `vc-manual.tex` to learn how the foot line has been set up in this document.

4. I want to have VCS data in the document only in draft mode!
Have a look at the `ifdraft` package from the `oberdiek` bundle.

5. How can I change the date format?

Package `isodate` provides means to convert between various (localized) date formats.

6. How can I access software specific VCS information, *e.g.*, Git's 40-hexdigit commit hash?

The general VCS information provided in macros prefixed `\VC` are available for all supported version control systems. However, there are additional information available for some VCS that are not for others. These information are stored in macros that have a VCS specific prefix (*cf.* appendix A and file `vc.tex`). As an example, Git's 40-hexdigit commit hash is provided in a macro `\GITHash`. You can either use this macro directly or redefine `\VCRevision` to show the long hash as follows:

```
\renewcommand*{\VCRevision}{\GITHash}
```

But keep in mind, that persons, unfamiliar with version control software or Git in particular, might be irritated by cryptic information on every document page.

7. Why are macros defined with `\gdef` instead of `\newcommand` in file `vc.tex`?

This is plain TEX syntax and this works with LATEX, too. If you're using LATEX you can of course use `\renewcommand` to redefine macros, *e.g.*, `\VCRevision` as shown in the answer to the preceeding question.

8. Macro `\VCRevisionMod` only shows the revision number, even if there are modified files in my working copy.

Call script `vc` with the `-m` switch, see appendix B.

9. In macro `\VCRevisionMod`, how can I get rid of the horizontal skip between revision number and the message?

By default, the definition of macro `\VCRevisionMod` is

```
\gdef\VCRevisionMod{\VCRevision~\VCModifiedText}
```

To remove the horizontal space before macro `\VCModifiedText` just start its definition with `\unskip`, *e.g.*,

```
\gdef\VCModifiedText{\unskip, modified}
```

10. How do I ignore files with my VCS?

If you don't know how to configure your VCS, please read its documentation carefully before doing any of the steps shown below!

Here are some short instructions for ignoring file `vc.tex` in Bazaar, Git, Mercurial and Subversion:

Bazaar In the directory containing the script files issue the following commands on the command line:

```
> bzr ignore vc.tex
> bzr commit .bzrignore
```

This creates a file `.bzrignore` that contains ignore patterns and puts that file under version control.

Git In the directory containing the script files create a file `.gitignore` containing the line

```
vc.tex
```

and put `.gitignore` under version control:

```
> git add .gitignore
> git commit .gitignore
```

Mercurial In the directory containing the script files create a file `.hgignore` containing the line

```
vc.tex
```

and put `.hgignore` under version control:

```
> hg add .hgignore
> hg commit .hgignore
```

Subversion In the directory containing the script files issue the following commands on the command line:

```
> svn propedit svn:ignore .
> svn commit
```

The first command will open an editor. Add the line

```
vc.tex
```

save the file, close the editor and commit the changes.

11. Is it possible to get per-file revision data with the `vc` bundle? This were quite handy when working with a multi-file document (say, each file is a chapter). After changing one file, the information could be used to check if an old print-out of another chapter is current.

This sounds like you're interested in tracking changes instead of just revision data. Note, that tracking changes and documenting revision data for later reference are fundamentally different requirements. The `vc` bundle has only been written with the latter use-case in mind. In fact, the scripts of the `vc` bundle try hard to be ignorant of individual file revision data.

There are three possible solutions (that do without `vc`):

1. To check whether files have changed between revisions one can use:

```
> svn diff -r ⟨r1⟩:⟨r2⟩ ⟨file⟩
```

Of course, this is an on-line only solution, while you can check printed numbers off-line and anywhere. That's not to say it's better or worse, but it requires a slightly different work-flow. (For example, to pass *anybody* the new document version, therefore avoiding the question if and where two print-outs differ.)

2. If you want to track changes instead of revision numbers, have a look at this item in UK-TeX-FAQ:
    - `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=changebars`
3. Packages `svninfo` or `svn-multi` and `vc` can perfectly be used together to get reliable total revision data as well as per-file revision data. Although, this may sound like overkill it could fit some use-cases. For alternative version control packages see
    - `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=RCS`
    - `http://texcatalogue.sarovar.org/bytopic.html#revision`

12. Can I put files of the `vc` bundle into a private, public or commercial repository?
This is perfectly possible. The `vc` bundle has been put into the Public Domain to remove any usage restrictions.

D Comparision with alternative VCS packages

The `vc` bundle

- looks at all files in a working copy to get reliable revision information,
- doesn't provide per-file revision data,
- doesn't use keyword substitution,
- works with LaTeX and plain TeX,
- supports Bazaar, Git, Mercurial and Subversion,
- needs an AWK interpreter.
- Running the scripts might become noticeable on projects with many files.

E To do

- Base Git scripts on plumbing commands.
- Rewrite (and merge) scripts in Perl.
- Add support for other VCS software. Contributions are welcome!