Excerpted from *bash Cookbook*

by Carl Albing, JP Vossen and Cameron Newham

For PANTUG, 2007-06-12

Book Timeline

2004-11-30 JP sends email to O'Reilly and proposes "How about a Bash cookbook?"

2005-03-24 Mike gets in touch

2005-04-01 Cameron (author of Learning the bash Shell) is added

2005-05-02 Submit second draft of ToC/outline

2005-0[89]* More drafts of the ToC/outline

2005-09-13 Carl is added

2005-10-01 Start learning/writing in OR wiki

2005-10* Submit first proposal, various revisions, contract discussion

2005-10-20 Get contract (600 pages, 12% of net revenue, split 3 ways more or less)

Schedule:

2005-11-30 3 chapters

2006-01-31 1/2 chapters

2006-05-31 First draft

2006-02-13 Got first advance check

2006-07-13 Move to OpenOffice.org Writer and deployed echidna (for SVN)

2006-12-18 Submit real first draft (for technical reviewers)

2006-12-18 Ship PDF draft (450 pages) to technical reviewers

2007-01-15 Technical Reviews due, give or take

2007-01-12 Got second/third advance check

2007-02-01 Submit copyediting draft

2007-02* Copyediting

2007-02-15 Got last advance check

2007-03-14 Draft from copyeditor to production

2007-03-27 Get PDFs for QC1

2007-03-29 Get FedEx of QC1 hardcopy (printouts of the QC1 PDFs)

2007-04-10 QC1 due back to O'Reilly

2007-04-25 Index (post QC1, pre QC2)

2007-05-03 QC2 review (PDF)

2007-05-11 Book goes to the printer!!!

D

Revision Control

Introduction

Revision control systems are a way to not only travel back in time, but to see what has changed at various points in your timeline. They are also called *versioning* or *version control* systems, which is actually a more technically accurate name. Such a system allows you to maintain a central *repository* of files in a project, and to keep track of changes to those files, as well as the reason for those changes. Some revision control systems allow more than one developer to work concurrently on the same project, or even the same file.

Revision control systems are essential to modern software development efforts, but they are also useful in many other areas, such as writing documentation, tracking system configurations (e.g., */etc*), and even writing books. We kept this book under revision control using Subversion while writing it.

Some of the useful features of revision control systems include:

- Making it very difficult to lose code, especially when the repository is properly backed up.
- Facilitating change control practices, and encourage documenting why a change is being made.
- Allowing people in multiple locations to work together on a project, and to keep up with others' changes, without losing data by saving on top of each other.
- Allowing one person to work from multiple locations over time without losing work or stepping on changes made at other locations.
- Allowing you to back out changes easily or to see exactly what has changed between one revision and another (except binary files). If you follow effective logging practices, they will even tell you why a change was made.
- Allowing, usually, a form of *keyword* expansion that lets you embed revision metadata in nonbinary files.

There are many different free and commercial revision control systems, and we would like to strongly encourage you to use one. If you already have one, use it. If you don't, we'll briefly cover three of the most common systems (CVS, Subversion, and RCS), all of which either come with or are available for every major modern operating system.

Before using a revision control system, you must first decide:

- Which system or product to use
- The location of the central repository, if applicable
- The structure of the projects or directories in the repository
- The update, commit, tag, and branch polices

This only scratches the surface; see O'Reilly's *Essential CVS* by Jennifer Vesperman and *Version Control with Subversion* by Ben Collins-Sussman et al. for more in-depth introductions to revision control and complete details on their respective systems. Both have excellent treatments of the general concepts, although the Subversion book covers repository structure in more detail due to its more fluid nature.

Both also cover revision control policy. If your company has change control or related policies, use them. If not, we recommend you commit and update early and often. If you are working as a team, we strongly recommend reading one or both of the books and carefully planning out a strategy. It will save vast amounts of time in the long run.

CVS

The Concurrent Versions System (CVS) is a widely used and mature revision control system, with command-line tools for all major modern operating systems (including Windows), and GUI tools for some of them (notably Windows).

Pros

- It is everywhere and is very mature.
- Many Unix system administrators and virtually every open source or free software developer is familiar with it.
- It's easy to use for simple projects.
- It's easy to access remote repositories.
- It's based on RCS, which allows for some hacking of the central repository.

Cons

- Commits are not atomic, so the repository could be left in an inconsistent state if a commit fails half-way through.
- Commits are by file only; you must also tag if you need to reference a group of files.
- Directory structure support is poor.
- Does not allow easy renaming of files and directories while retaining history.
- Poor support for binary files, and little support for other objects such as symbolic links.
- Based on RCS, which allows for some hacking of the central repository.

CVS tracks revisions by file, which means that each file has its own internal CVS revision number. As each file is changed, that number changes, so a single project can't be tracked by a single revision

---

number, since each file is different. Use tags for that kind of tracking.

Example

This example is not suitable for enterprise or multiuser access (see the references provided in the “See Also” section below for that). This is just to show how easy the basics are. This example has the `EDITOR` environment variable set to *nano* (`export EDITOR='nano --smooth --const --nowrap --suspend'`), which some people find more user-friendly than the default *vi*.

The `cvs` command (with no options), the `cvs help` command (where `help` is not a valid argument, but is easy to remember and still triggers a useful response), and the `cvs --help` *`cvs_command`* command are very useful.

Create a new repository for personal use in a home directory:

```
/home/jp$ mkdir -m 0775 cvsroot
/home/jp$ chmod g+srwx cvsroot
/home/jp$ cvs -d /home/jp/cvsroot init
```

Create a new project and import it:

```
/home/jp$ cd /tmp

/tmp$ mkdir -m 0700 scripts

/tmp$ cd scripts/

/tmp/scripts$ cat << EOF > hello
> #!/bin/sh
> echo 'Hello World!'
> EOF

/tmp/scripts$ cvs -d /home/jp/cvsroot import scripts shell_scripts NA

  GNU nano 1.2.4                      File: /tmp/cvsnJgYmG

Initial import of shell scripts
CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS:' are removed automatically
CVS:
CVS: ----------------------------------------------------------------------

                                      [ Wrote 5 lines ]

N scripts/hello

No conflicts created by this import
```

Check out the project and update it:

```
/tmp/scripts$ cd
/home/jp$ cvs -d /home/jp/cvsroot/ checkout scripts
cvs checkout: Updating scripts
U scripts/hello

/home/jp$ cd scripts

/home/jp/scripts$ ls -l
total 8.0K
drwxr-xr-x  2 jp jp 4.0K Jul 20 00:27 CVS/
-rw-r--r--  1 jp jp   30 Jul 20 00:25 hello

/home/jp/scripts$ echo "Hi Mom..." >> hello
```

Check the status of your sandbox. The second command is a hack to give you a short summary status since the real status command is a little verbose:

```
/home/jp/scripts$ cvs status
cvs status: Examining .
===================================================================
File: hello             Status: Locally Modified

   Working revision:    1.1.1.1 Thu Jul 20 04:25:44 2006
   Repository revision: 1.1.1.1 /home/jp/cvsroot/scripts/hello,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

/home/jp/scripts$ cvs -qn update
M hello
```

Add a new script to revision control:

```
/home/jp/scripts$ cat << EOF > mcd
> #!/bin/sh
> mkdir -p "$1"
> cd "$1"
> EOF

/home/jp/scripts$ cvs add mcd
cvs add: scheduling file `mcd' for addition
cvs add: use `cvs commit' to add this file permanently
```

Commit changes:

```
/home/jp/scripts$ cvs commit
cvs commit: Examining .

  GNU nano 1.2.4                      File: /tmp/cvsY1xcKa

* Tweaked hello
* Added mcd
CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:    hello
CVS: Added Files:
CVS:    mcd
CVS: ----------------------------------------------------------------------

                                     [ Wrote 12 lines ]

/home/jp/cvsroot/scripts/hello,v  <--  hello
new revision: 1.2; previous revision: 1.1
/home/jp/cvsroot/scripts/mcd,v  <--  mcd
initial revision: 1.1
```

Update the sandbox, make another change, then check the difference:

```
/home/jp/scripts$ cvs update
cvs update: Updating .

/home/jp/scripts$ vi hello

/home/jp/scripts$ cvs diff hello
Index: hello
===================================================================
RCS file: /home/jp/cvsroot/scripts/hello,v
```

```
retrieving revision 1.2
diff -r1.2 hello
3c3
< Hi Mom...
---
> echo 'Hi Mom...'
```

Commit the change, avoiding the editor by putting the log entry on the command line:

```
/home/jp/scripts$ cvs -m '* Fixed syntax error' commit
/home/jp/cvsroot/scripts/hello,v  <--  hello
new revision: 1.3; previous revision: 1.2
```

See the history of the file:

```
/home/jp/scripts$ cvs log hello

RCS file: /home/jp/cvsroot/scripts/hello,v
Working file: hello
head: 1.3
branch:
locks: strict
access list:
symbolic names:
        NA: 1.1.1.1
        shell_scripts: 1.1.1
keyword substitution: kv
total revisions: 4;     selected revisions: 4
description:
----------------------------
revision 1.3
date: 2006-07-20 04:46:25 +0000;  author: jp;  state: Exp;  lines: +1 -1
* Fixed syntax error
----------------------------
revision 1.2
date: 2006-07-20 04:37:37 +0000;  author: jp;  state: Exp;  lines: +1 -0
* Tweaked hello
* Added mcd
----------------------------
revision 1.1
date: 2006-07-20 04:25:44 +0000;  author: jp;  state: Exp;
branches:  1.1.1;
Initial revision
----------------------------
revision 1.1.1.1
date: 2006-07-20 04:25:44 +0000;  author: jp;  state: Exp;  lines: +0 -0
Initial import of shell scripts
=============================================================================
```

Add some revision metadata that is automatically kept up-to-date by the revision control system itself. Commit it and examine the change:

```
/home/jp/scripts$ vi hello

/home/jp/scripts$ cat hello
#!/bin/sh
# $Id$
echo 'Hello World!'
echo 'Hi Mom...'

/home/jp/scripts$ cvs ci -m'* Added ID keyword' hello
/home/jp/cvsroot/scripts/hello,v  <--  hello
new revision: 1.4; previous revision: 1.3

/home/jp/scripts$ cat hello
```

```
#!/bin/sh
# $Id: hello,v 1.4 2006/07/21 08:57:53 jp Exp $
echo 'Hello World!'
echo 'Hi Mom...'
```

Compare the current revision to r1.2, revert to that older (broken) revision, realize we goofed and get the most recent revision back:

```
/home/jp/cvs.scripts$ cvs diff -r1.2 hello
Index: hello
===================================================================
RCS file: /home/jp/cvsroot/scripts/hello,v
retrieving revision 1.2
retrieving revision 1.4
diff -r1.2 -r1.4
1a2
> # $Id: hello,v 1.4 2006/07/21 08:57:53 jp Exp $
3c4
< Hi Mom...
---
> echo 'Hi Mom...'

/home/jp/scripts$ cvs update -r1.2 hello
U hello

/home/jp/scripts$ cat hello
#!/bin/sh
echo 'Hello World!'
Hi Mom...

/home/jp/cvs.scripts$ cvs update -rHEAD hello
U hello

/home/jp/cvs.scripts$ cat hello
#!/bin/sh
# $Id: hello,v 1.4 2006/07/21 08:57:53 jp Exp $
echo 'Hello World!'
echo 'Hi Mom...'
```

See Also

[Recipe #Creating and Changing Into a New Directory in One Step]

man cvs

man rcs2log

man cvs-pserver

Essential CVS, Second Edition

The official CVS web site, at http://www.nongnu.org/cvs/

CVS Docs and Cederqvist manual, at http://ximbiot.com/cvs/manual/

Windows shell extension for CVS, at http://www.tortoisecvs.org/

“Introduction to CVS,” at http://linux.oreillynet.com/lpt/a/1420

“CVS Administration,” at http://linux.oreillynet.com/lpt/a/1421

“Tracking Changes in CVS,” at http://linux.oreillynet.com/lpt/a/2443

“CVS Third-Party Tools,” at http://www.onlamp.com/lpt/a/2895

“Top 10 CVS Tips,” at http://www.oreillynet.com/lpt/a/2015

“CVS Branch and Tag Primer,” at http://www.psc.edu/~semke/cvs_branches.html

“CVS Best Practices,” at http://www.tldp.org/REF/CVS-BestPractices/html/index.html

Subversion

According to the Subversion web site, “The goal of the Subversion project is to build a *version control system* that is a compelling replacement for CVS in the open source community.“ Enough said.

Pros

- Newer than CVS and RCS.
- Simpler and arguably easier to understand and use than CVS (less historical baggage).
- Atomic commits means the commit either fails or succeeds as a whole, and makes it easy to track the state of an entire project as a single revision.
- Easy to access remote repositories.
- Allows easy renaming of files and directories while retaining history.
- Easily handles binary files (no native diff support) and other objects such as symbolic links.
- Central repository hacking is more officially supported, but less trivial.

Cons

- Not 100 percent CVS compatible for more complicated projects (e.g., branching and tagging).
- Can be more complicated to build or install from scratch due to many dependencies. Use the version that came with your operating system if possible.

> SVN tracks revisions by repository, which means that each commit has its own internal SVN revision number. Thus consecutive commits by a single person may not have consecutive revision numbers since the global repository revision is incremented as other changes (possibly to other projects) are committed by other people.

Example

This example is not suitable for enterprise or multiuser access (see the references provided in the “See Also” section below for that). This is just to show how easy the basics are. This example also has the `EDITOR` environment variable set to *nano* (`export EDITOR='nano --smooth --const --nowrap --suspend'`), which some people find more user-friendly than the default *vi*.

The `svn help` and `svn help help` commands are very useful.

Create a new repository for personal use in a home directory:

```
/home/jp$ svnadmin --fs-type=fsfs create /home/jp/svnroot
```

Create a new project and import it:

```
/home/jp$ cd /tmp

/tmp$ mkdir -p -m 0700 scripts/trunk scripts/tags scripts/branches
```

```
/tmp$ cd scripts/trunk

/tmp/scripts/trunk$ cat << EOF > hello
> #!/bin/sh
> echo 'Hello World!'
> EOF

/tmp/scripts/trunk$ cd ..

/tmp/scripts$ svn import /tmp/scripts file:///home/jp/svnroot/scripts

  GNU nano 1.2.4                      File: svn-commit.tmp

Initial import of shell scripts
--This line, and those below, will be ignored--

A    .

                                      [ Wrote 4 lines ]

Adding         /tmp/scripts/trunk
Adding         /tmp/scripts/trunk/hello
Adding         /tmp/scripts/branches
Adding         /tmp/scripts/tags

Committed revision 1.
```

Check out the project and update it:

```
/tmp/scripts$ cd

/home/jp$ svn checkout file:///home/jp/svnroot/scripts
A  scripts/trunk
A  scripts/trunk/hello
A  scripts/branches
A  scripts/tags
Checked out revision 1.

/home/jp$ cd scripts

/home/jp/scripts$ ls -l
total 12K
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 branches/
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 tags/
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 trunk/

/home/jp/scripts$ cd trunk/

/home/jp/scripts/trunk$ ls -l
total 4.0K
-rw-r--r--  1 jp jp 30 Jul 20 01:12 hello

/home/jp/scripts/trunk$ echo "Hi Mom..." >> hello
```

Check the status of your sandbox. Note how the `svn status` command is similar to our `cvs -qn update` hack in the “CVS” section earlier in this appendix:

```
/home/jp/scripts/trunk$ svn info
Path: .
URL: file:///home/jp/svnroot/scripts/trunk
Repository UUID: 29eeb329-fc18-0410-967e-b075d748cc20
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: jp
```

```
Last Changed Rev: 1
Last Changed Date: 2006-07-20 01:04:56 -0400 (Thu, 20 Jul 2006)

/home/jp/scripts/trunk$ svn status -v
                1        1 jp           .
M               1        1 jp           hello

/home/jp/scripts/trunk$ svn status
M      hello

/home/jp/scripts/trunk$ svn update
At revision 1.
```

Add a new script to revision control:

```
/home/jp/scripts/trunk$ cat << EOF > mcd
> #!/bin/sh
> mkdir -p "$1"
> cd "$1"
> EOF

/home/jp/scripts/trunk$ svn st
?      mcd
M      hello

/home/jp/scripts/trunk$ svn add mcd
A         mcd
```

Commit changes:

```
/home/jp/scripts/trunk$ svn ci

  GNU nano 1.2.4                      File: svn-commit.tmp

* Tweaked hello
* Added mcd
--This line, and those below, will be ignored--

M    trunk/hello
A    trunk/mcd

                                      [ Wrote 6 lines ]

Sending        trunk/hello
Adding         trunk/mcd
Transmitting file data ..
Committed revision 2.
```

Update the sandbox, make another change, then check the difference:

```
/home/jp/scripts/trunk$ svn up
At revision 2.

/home/jp/scripts/trunk$ vi hello

/home/jp/scripts/trunk$ svn diff hello
Index: hello
===================================================================
--- hello       (revision 2)
+++ hello       (working copy)
@@ -1,3 +1,3 @@
 #!/bin/sh
 echo 'Hello World!'
-Hi Mom...
+echo 'Hi Mom...'
```

Commit the change, avoiding the editor by putting the log entry on the command

line:

```
/home/jp/scripts/trunk$ svn -m '* Fixed syntax error' commit
Sending        trunk/hello
Transmitting file data .
Committed revision 3.
```

See the history of the file:

```
/home/jp/scripts/trunk$ svn log hello
------------------------------------------------------------------------
r3 | jp | 2006-07-20 01:23:35 -0400 (Thu, 20 Jul 2006) | 1 line

* Fixed syntax error
------------------------------------------------------------------------
r2 | jp | 2006-07-20 01:20:09 -0400 (Thu, 20 Jul 2006) | 3 lines

* Tweaked hello
* Added mcd

------------------------------------------------------------------------
r1 | jp | 2006-07-20 01:04:56 -0400 (Thu, 20 Jul 2006) | 2 lines

Initial import of shell scripts

------------------------------------------------------------------------
```

Add some revision metadata, and tell the system to expand it. Commit it and examine the change:

```
/home/jp/scripts/trunk$ vi hello

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id$
echo 'Hello World!'
echo 'Hi Mom...'

home/jp/scripts/trunk$ svn propset svn:keywords "Id" hello
property 'svn:keywords' set on 'hello'

/home/jp/scripts/trunk$ svn ci -m'* Added ID keyword' hello
Sending        hello

Committed revision 4.

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id: hello 5 2006-07-21 09:09:34Z jp $
echo 'Hello World!'
echo 'Hi Mom...'
```

Compare the current revision to r2, revert to that older (broken) revision, realize we goofed and get the most recent revision back:

```
/home/jp/scripts/trunk$ svn diff -r2 hello
Index: hello
===================================================================
--- hello       (revision 2)
+++ hello       (working copy)
@@ -1,3 +1,4 @@
 #!/bin/sh
+# $Id$
 echo 'Hello World!'
-Hi Mom...
+echo 'Hi Mom...'
```

```
Property changes on: hello
___________________________________________________________________
Name: svn:keywords
   + Id

/home/jp/scripts/trunk$ svn update -r2 hello
UU hello
Updated to revision 2.

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
echo 'Hello World!'
Hi Mom...

/home/jp/scripts/trunk$ svn update -rHEAD hello
UU hello
Updated to revision 4.

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id: hello 5 2006-07-21 09:09:34Z jp $
echo 'Hello World!'
echo 'Hi Mom...'
```

See Also

[Recipe #Creating and Changing Into a New Directory in One Step]

man svn

man svnadmin

man svndumpfilter

man svnlook

man svnserve

man svnversion

The Subversion web site, at http://subversion.tigris.org/

TortoiseSVN: Simple SVN front-end for Explorer (Cool!), at http://tortoisesvn.tigris.org/

Version Control with Subversion, at http://svnbook.red-bean.com/

SVN static builds for solaris, linux, mac, at http://www.uncc.org/svntools/clients/

“Subversion for CVS Users,” at http://osdir.com/Article203.phtml

Version control system comparison, at http://better-scm.berlios.de/comparison/comparison.html

RCS

RCS was a revolution in its time, and is the underlying basis for CVS.

Pros

- It's better than nothing.

Cons

- Does not allow concurrent access to the same file.
- Does not have the inherent concept of a central repository, though you can go out of your way to create one using symbolic links.
- No concept of remote repositories.
- Only tracks changes to files, and does not store or consider directories at all.
- Poor support for binary files, and no support for other objects such as symbolic links.Unlike CVS or SVN, which have a single main end-user binary, RCS is a collection of binaries.

Example

Create a new script directory for personal use in a home directory:

```
/home/jp$ mkdir -m 0754 bin
```

Create some scripts:

```
/home/jp$ cd bin

/tmp/scripts/bin$ cat << EOF > hello
> #!/bin/sh
> echo 'Hello World!'
> EOF

/home/jp/bin$ ci hello
hello,v  <--  hello
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Obligatory Hello World
>> .
initial revision: 1.1
done

/home/jp/bin$ ls -l
total 4.0K
-r--r--r--  1 jp jp 228 Jul 20 02:25 hello,v
```

Huh? What happened? It turns out that if a directory called *RCS* does not exist, the current directory is used for the RCS file. And if the `-u` or `-l` switches are not used, the file is checked in and then removed. `-l` causes the file to be checked back out and locked so you can edit it, while `-u` is unlocked (that is, read-only). OK, let's try that again. First, let's get our file back, then create an *RCS* directory and check it in again.

```
/home/jp/bin$ co -u hello
hello,v  -->  hello
revision 1.1 (unlocked)
done

/home/jp/bin$ ls -l
total 8.0K
-r--r--r--  1 jp jp  30 Jul 20 02:29 hello
-r--r--r--  1 jp jp 228 Jul 20 02:25 hello,v

/home/jp/bin$ rm hello,v
rm: remove write-protected regular file `hello,v'? y

/home/jp/bin$ mkdir -m 0755 RCS
```

```
/home/jp/bin$ ci -u hello
RCS/hello,v  <--  hello
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Obligatory Hello World
>> .
initial revision: 1.1
done

/home/jp/bin$ ls -l
total 8.0K
drwxr-xr-x  2 jp jp 4.0K Jul 20 02:31 RCS/
-r--r--r--  1 jp jp   30 Jul 20 02:29 hello

/home/jp/bin$ ls -l RCS
total 4.0K
-r--r--r--  1 jp jp 242 Jul 20 02:31 hello,v
```

Note that our original file is now read-only. This is to remind us to check it out using `co -l` before working on it. Let's do that:

```
/home/jp/bin$ co -l hello
RCS/hello,v  -->  hello
revision 1.1 (locked)
done

/home/jp/bin$ ls -l
total 8.0K
drwxr-xr-x  2 jp jp 4.0K Jul 20 02:39 RCS/
-rw-r--r--  1 jp jp   30 Jul 20 02:39 hello

/home/jp/bin$ echo "Hi Mom..." >> hello
```

Commit changes, but keep a copy locked for editing:

```
/home/jp/bin$ ci -l hello
RCS/hello,v  <--  hello
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> * Tweaked hello
>> .
done

/home/jp/bin$ ls -l
total 8.0K
drwxr-xr-x  2 jp jp 4.0K Jul 20 02:44 RCS/
-rw-r--r--  1 jp jp   40 Jul 20 02:39 hello
```

Make another change, then check the difference:

```
/home/jp/bin$ vi hello

/home/jp/bin$ rcsdiff hello
===================================================================
RCS file: RCS/hello,v
retrieving revision 1.2
diff -r1.2 hello
3c3
< Hi Mom...
---
> echo 'Hi Mom...'
```

Commit the change, and keep an unlocked copy for actual use:

```
/home/jp/bin$ ci -u -m'* Fixed syntax error' hello
RCS/hello,v  <--  hello
```

```
new revision: 1.3; previous revision: 1.2
done

/home/jp/bin$ ls -l
total 8.0K
drwxr-xr-x  2 jp jp 4.0K Jul 20 02:46 RCS/
-r--r--r--  1 jp jp   47 Jul 20 02:45 hello
```

See the history of the file:

```
/home/jp/bin$ rlog hello

RCS file: RCS/hello,v
Working file: hello
head: 1.3
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 3;     selected revisions: 3
description:
Obligatory Hello World
----------------------------
revision 1.3
date: 2006/07/20 06:46:30;  author: jp;  state: Exp;  lines: +1 -1
* Fixed syntax error
----------------------------
revision 1.2
date: 2006/07/20 06:43:54;  author: jp;  state: Exp;  lines: +1 -0
* Tweaked hello
----------------------------
revision 1.1
date: 2006/07/20 06:31:06;  author: jp;  state: Exp;
Obligatory Hello World
=============================================================================
```

Add some revision metadata, and tell the system to expand it. Commit it and examine the change:

```
/home/jp/bin$ co -l hello
RCS/hello,v  -->  hello
revision 1.3 (locked)
done

/home/jp/bin$ vi hello

/home/jp/bin$ cat hello
#!/bin/sh
# $Id$
echo 'Hello World!'
echo 'Hi Mom...'

/home/jp/bin$ ci -u -m'Added ID keyword' hello
RCS/hello,v  <--  hello
new revision: 1.4; previous revision: 1.3
done

/home/jp/bin$ cat hello
#!/bin/sh
# $Id: hello,v 1.4 2006/07/21 09:18:09 jp Exp $
echo 'Hello World!'
echo 'Hi Mom...'
```

Compare the current revision to r1.2, revert to that older (broken) revision, realize we goofed and get the most recent revision back:

```
/home/jp/bin$ rcsdiff -r1.2 hello
===================================================================
RCS file: RCS/hello,v
retrieving revision 1.2
diff -r1.2 hello
1a2
> # $Id: hello,v 1.4 2006/07/21 09:18:09 jp Exp $
3c4
< Hi Mom...
---
> echo 'Hi Mom...'

/home/jp/bin$ co -r hello
RCS/hello,v  -->  hello
revision 1.4
writable hello exists; remove it? [ny](n): y
done

/home/jp/bin$ cat hello
#!/bin/sh
# $Id: hello,v 1.4 2006/07/21 09:18:09 jp Exp $
echo 'Hello World!'
echo 'Hi Mom...'
```

Workon Script

Here is a script that may make life with RCS a little easier. It facilitates using an RCS "repository" and automates much of the process of checking files in and out to work on them, hence the name. We recommend that you use Subversion or CVS if possible, but if you must use RCS you may find this helpful:

```
#!/usr/bin/env bash
# cookbook filename: workon
# workon--Work on a file in RCS

# Set a sane/secure path and export it
PATH=/usr/local/bin:/bin:/usr/bin
export PATH

VERSION='$Version: 1.4 $' # JP Vossen
COPYRIGHT='Copyright 2004-2006 JP Vossen (http://www.jpsdomain.org/)'
LICENSE='GNU GENERAL PUBLIC LICENSE'

CAT='/bin/cat'
if [ "$1" = "-h" -o "$1" = "--help" -o -z "$1" ]; then
    ${CAT} <<-EoN
    Usage: $0 {file}

    Work on a file in RCS.  Create the RCS subdirectory if necessary.
    Do the initial checkin if necessary, prompting for a message.
    Must be in the same directory as the file to be worked on.
EoN
    exit 0
fi

# Use a pseudo central repository
RCSHOMEDIR='/home/rcs'

# Make sure $VISUAL is set to something
[ "$VISUAL" ] || VISUAL=vi

########################################################################
# Start of Main program
```

```
# Make sure RCS Home Dir exists
if [ ! -d $RCSHOMEDIR ]; then
    echo "Creating $RCSHOMEDIR..."
    mkdir -p $RCSHOMEDIR
fi

# Make sure there is no local RCS directory
if [ -d RCS -a ! -L RCS ]; then
    echo "Local 'RCS' already exists--exiting!"
    exit 2
fi

# Make sure the destdir exists
if [ ! -d $RCSHOMEDIR$PWD ]; then
    echo "Creating $RCSHOMEDIR$PWD..."
    mkdir -p $RCSHOMEDIR$PWD
fi

# Make sure the link exists
if [ ! -L RCS ]; then
    echo "Linking RCS --> $RCSHOMEDIR$PWD."
    ln -s $RCSHOMEDIR$PWD RCS
fi

if [ ! -f "RCS/$1,v" ]; then
    # If the file is not ALREADY in RCS add it as v1.0.

    echo 'Adding "Initial Revision/Default" of file to RCS...'

    # Get input
    echo -n 'Describe this file: '
    read logmsg

    # Check in v1.0
    ci -u1.0 -t-"$logmsg" -m'Initial Revision/Default' $1

else
    # If the file is in RCS, work on it.

    # Checkout the file in locked mode for editing
    co -l $1

    # Edit the file locally
    $VISUAL $1

    # Check the file back in, but keep a read-only copy out for use
    ci -u $1
fi
```

See Also

man ci

man co

man ident

man merge

man rcs

man rcsclean

man rcsdiff

man rcsmerge

man rlog

man rcsfreeze

Applying RCS and SCCS

“BSD Tricks: Introductory Revision Control,” at http://www.onlamp.com/lpt/a/428

Applying RCS and SCCS, Chapter 3: Basic Source Control Using RCS, at http://www.oreilly.com/catalog/rcs/chapter/ch03.html

Other

Finally, it is worth noting that some word processors, such as OpenOffice.org Writer and Microsoft Word, have three relevant features: document comparison, change tracking, and versions.

Document Comparison

Document Comparison allows you to compare documents when their native file format makes use of other *diff* tools difficult. You would use this when you have two copies of a document that didn't have change tracking turned on, or when you need to merge feedback from various sources.

While it is trivial to *unzip* the *content.xml* file from a given OpenDoc file, the result has no line breaks and is not terribly pretty or readable. See [Recipe #Comparing Two Documents] in Chapter 12 for a *bash* script that will do this low-level kind of difference.

Refer to the table below for information on how to access the built-in GUI comparison function, which is much easier than trying to do it manually.

Change Tracking and Versions

The change-tracking feature saves information about changes made to a document. Review mode uses various copyediting markup on the screen to display who did what, when. This is obviously useful for all kinds of creation and editing purposes, but please read our warnings.

The versions feature allows you to save more than one version of a document in a single file. This can be handy in all sorts of odd ways. For example, we've seen router configurations copied and pasted from a terminal into different versions inside the same document for archival and change control purposes.

> The change tracking and versions features will cause your document to continually grow in size, since items that are changed are still kept and deleted items are not really deleted, but only marked as deleted.

> If accidentally turned on, change tracking and versions can be very dangerous information leaks! For example, if you send similar proposals to competing companies after doing a search and replace and other editing, someone at one of those companies can see exactly what you changed and when you changed it. The most recent versions of these tools have various methods that attempt to warn you or clear private information before a given document is converted to PDF or emailed.
>
> Take a look at any word processor attachments you receive in email,

especially from vendors. You may be surprised. For an excellent example of this, see “Politics: Documents Reveal US Incompetence with Word, Iraq” at http://politics.slashdot.org/politics/07/05/18/132235.shtml and http://www.salon.com/news/feature/2007/05/18/cpa_documents/.

http://politics.slashdot.org/politics/07/05/18/132235.shtml

Politics: Documents Reveal US Incompetence with Word, Iraq
Posted by Zonk on Friday May 18, @09:27AM
from the see-there's-this-thing-called-covering-your-tracks dept.

notNeilCasey writes "The U.S. Coalition Provisional Authority, which formerly governed Iraq, accidentally published Microsoft Word documents containing information never meant for the public, according to an article in Salon. By viewing the documents using the Track Changes feature in Word (.doc), the author has been able to reconstruct internal discussions from 2004 which reflect the optimism, isolation and incompetence of the American occupation. Download the author's source document or look for more yourself. 'Presumably, staffers at the CPA's Information Management Unit, which produced the weekly reports, were cutting and pasting large sections of text into the reports and then eliminating all but the few short passages they needed. Much of the material they were cribbing seems to have come from the kind of sensitive, security-related documents that were never meant to be available to the public. In fact, about half of the 20 improperly redacted documents I downloaded, including the March 28 report, contain deleted portions that all seem to come from one single, 1,000-word security memo. The editors kept pulling text from a document titled "Why Are the Attacks Down in Al-Anbar Province -- Several Theories." (The security memo and the last page of the March 28 report can be seen here, along with several other CPA documents that can be downloaded.)'"

http://www.salon.com/news/feature/2007/05/18/cpa_documents/

http://media.salon.com/doc/Administrators_WEEKLY_ECONOMIC_REPORT_March28-2004a.doc

http://www.cpa-iraq.org/

Accessing These Features

Feature	Writer menu option	Word menu option
Document comparisons	Edit→Compare Document	Tools→Compare and Merge Documents
Change tracking	Edit→Changes	Tools→Track Changes
Versions	File→Versions	File→Versions