toolman

Revision Control Revisited

You've probably seen articles extolling the virtues of revision-control systems. I'm not about to argue with their premise, nor am I going to repeat the discussion. What I will talk about here is a method of automating what is probably the most common sequence of actions associated with revision control: the cycle of check-out, edit, check-in. Presumably, if this process is made easier, use of revision control in appropriate situations would become more likely; we'll assume that this is a good thing. Also, if this process is more automated and reliable, then errors are presumably less likely (for example, forgetting to check files back in, or editing as root); this is also a good thing.

by Daniel E. Singer

Dan has been doing a mix of programming and system administration since 1983. He is currently a system administrator in the Duke University Department of Computer Science in Durham, North Carolina, USA.

<des@cs.duke.edu>

Background

In the UNIX world, the two most common packages used for revision control are SCCS (Source Code Control System) and RCS (Revision Control System). I like the former a little better, because I perceive it as being a bit easier to use (probably just a case of familiarity). But I've heard it said that the latter is more powerful, though that doesn't matter much to me since my usage pattern is very simple: I'm just interested in checking out a single file at a time, editing it, and checking it back in.[1] Also, SCCS is subject to licensing restrictions, though it is bundled with many UNIXes, while RCS is freely available and is also the basis for CVS (Concurrent Versions System). Various other implementations of revision-control systems are also available, both commercially and otherwise, and with varying degrees of application specificity; see the Resources section below.

Typical Command Sequences

The most common sequence of actions used with revision-control systems (for me, at least) is: check-out, edit, check-in. With SCCS, this will often be done with commands like those below (italics indicates user keystrokes):

```
% sccs edit filename
% vi filename
% sccs delget filename
```

Some variations are possible. Also, your editor religion, er, choice may vary.

With RCS, commands such as these might typically be used:

```
% co -l filename
% vi filename
% ci -u filename
```

Again, variations are possible, even likely. Please see the SCCS and RCS `man` pages for details and subtleties on command syntax, usage, and options.

The above commands look easy (and they are), but they're often a lot of typing for a few little changes, and errors are possible. To put it another way, when you have to use them over and over again, they're tedious and a pain in the butt.

The Tool

To make the check-out, edit, check-in cycle a little more bearable, I composed a tool some years ago to automate this process, and gradually refined it and added various conveniences, sanity checks, and features as needed over time. For instance, it will first tell you if anything in the directory is already checked-out and will warn you if it detects that you are root-ly (as it is generally the custom not to intentionally modify

There are several opportunities to bail out, and you can even have it “unedit” the file.

revision-controlled files as root). Also, I first wrote it for just SCCS (recall my bias), but I later added RCS capability for a colleague who preferred that.

Here’s a screen listing of a run of the script:

```
% rc usenix-article <RETURN>
rc: Using editor "vi"...
rc: sccs check
rc: Nothing being edited.

Continue (y/n)? [y] <RETURN>
rc: Checking out "usenix-article"...
rc: sccs edit "usenix-article"
1.7
new delta 1.8
679 lines

rc: Edit "usenix-article" (y/n)? [y] <RETURN>
[... edit session here ...]

rc: Check in "usenix-article" (y/n)? [y] <RETURN>
rc: Checking in "usenix-article"...

*** COMMENTS ***

rc: sccs delget "usenix-article"
comments? revised the part about revisions <RETURN>
1.8
27 inserted
27 deleted
652 unchanged
1.8
679 lines
rc: Done.
%
```

Notice that other than typing the single command (with the very short command name), editing, adding the usual comment, and hitting the return key a few times, there was nothing else to type. There are several opportunities to bail out (or just type Control-C), and you can even have it “unedit” (in SCCS lingo) the file. Here’s an example of that:

```
% rc usenix-article <RETURN>
rc: Using editor "vi"...
rc: sccs check
rc: Nothing being edited.

Continue (y/n)? [y] <RETURN>
rc: Checking out "usenix-article"...
rc: sccs edit "usenix-article"
1.8
new delta 1.9
679 lines

rc: Edit "usenix-article" (y/n)? [y] <RETURN>
[... edit session here ...]

rc: Check in "usenix-article" (y/n)? [y] n
rc: Unedit "usenix-article" (y/n)? [n] y
rc: Unediting "usenix-article"...
rc: sccs unedit "usenix-article"
usenix-article: removed
1.8
679 lines
rc: Done.

%
```

If the file you want to check out is already checked-out, `rc` (Revision Control) will warn you and ask if you want to continue. If you do continue, it will skip the check-out and go straight to the edit step.

If the file does not yet exist, `rc` will warn you and ask if you want to continue. If you do continue, it again will go straight to the edit step, and will then ask if you want to create a history file (that is, check it in after editing is done).

Preference

As mentioned, `rc` will work with both SCCS and RCS. In fact, it will do its pointy-headed best to figure out which to use in a given situation, using a more or less heuristic approach; that is, it takes its best guess.

In an ambiguous situation, or when the file does not already exist, `rc` will employ a preference that you can set either with a variable or a command-line option. The variable `RC_FAVOR` can be set in the script itself or in the environment to be one of `SCCS` or `RCS`. This can be overridden by supplying either the `-s` or `-r` command-line option.

In an ambiguous situation, or when the file does not already exist, `rc` will employ a preference that you can set either with a variable or a command-line option.

Executabits

One annoyance of SCCS (RCS is immune to this one) is the nonretention of the execute bits in the file permissions. That is, when you check a file out with SCCS, the working copy loses any execute bits that were set, and when you check the file back in, they stay lost. This is inconvenient, say, if your file is a script, and the working copy happens to also be the actual copy that gets used in real life.[2] So, if for instance you have a file `welcome-to-our-site.cgi`, and you do a check-out, edit, check-in, and you forget to do the pesky `chmod` to reenable the execute bits, then your whole Web site is hosed, and you just lost your job, and you can no longer feed your family, and . . .

`rc` will track and set the execute bits both during the file check-out and later during the file check-in. Your family is safe!

Sleight-of-Hand

Since the editor used by `rc` can be determined by an environment variable, it's possible to have some action performed other than an interactive edit session.[3] For instance, another script or program could instead be invoked to perform some automated update (and possibly also call up an interactive editor), with `rc` again serving as a wrapper to deal with the revision control before and after. (This is why I added the "Using editor" message at the beginning of the script output.) An outer wrapper script might be used to accomplish this; for example:

```
#!/bin/sh
VISUAL="wizbang -x" rc $*
```

The next section describes an alternative approach to this.

Embeddability

I recently added some options to `rc` to make it more amenable to being used from within other scripts. I've been working on some code to implement a simple but flexible database system all via shell[4], and ran into an actual situation in which the database file that I wanted to somewhat transparently update was under SCCS. I wanted to incorporate some of the sanity checks and SCCS/RCS flexibility from `rc` but didn't want to try to add in all of that code. So I added to `rc` a "quiet" option and also options to isolate the check-out and check-in portions as separate invocations. After handling the updates in memory, the database script calls `rc -oq dbfile` to check out the file, then updates the file, then calls `rc -iq -c` "some comment" `dbfile` to check it back in.

rc is just plain simpler to use than SCCS or RCS directly, and this is especially true for naive users.

Thus, rc can not only be used as an interactive, command-line tool, but can also be embedded in other applications.

Pathnames

Another goodie provided by rc is the ability to specify your file with a full pathname. For example:

```
% rc ~/public_html/recipes/guacamole.html
```

No need to cd to the directory, modify the file, and then cd back, in those situations where you only need to adjust the coriander.

Simplicity

rc is just plain simpler to use than SCCS or RCS directly, and this is especially true for naive users. Let's say you've got a secretary who needs to occasionally maintain some files that are under revision control. He doesn't really need to be bothered with the intricacies of revision-control systems. You can just tell him, "Look, when you need to edit one of these files, just use this single, simple command. . . ."

Summary

So there you have it, a tool you can use to automate common usages of SCCS and RCS, reduce errors, and make life easier for both naive and sophisticated users.

Resources

On UNIX systems, see the online reference manual (the man pages) for sccs and rcs, to get all of the options and subcommands that are supported on your system.

RCS and CVS are available through the Free Software Foundation. See these Web pages for more information:

<http://www.fsf.org/software/rcs/>
<http://www.fsf.org/software/cvs/>

The book on SCCS and RCS:

Bolinger, Don, and Tan Bronson. *Applying RCS and SCCS*. Sebastopol, CA: O'Reilly & Associates, 1995. <http://www.oreilly.com/catalog/rcs/>

Nick Christenson reviews the Bolinger and Bronson book glowingly in the December 1998 issue of *;login:* at <http://www.usenix.org/publications/login/1998-12/rcs.html>.

A good introductory article on RCS, and a pair on CVS:

Copeland, Jeffrey, and Jeffrey Haemer. "Practical RCS," "Practical CVS, Part 1," and "Practical CVS, Part 2." *Server/Workstation Expert* July, August, September 1997. <http://sw.expert.com/>

An article on how to apply this stuff more directly to our trade:

LeFebvre, William. "Revision Control for System Administrators." *Performance computing*, May 1998. <http://www.performancecomputing.com/>

A good place to find information about several revision-control systems is the Cyclic Software Web site, <http://www.cyclic.com/gallery/index.html>.

Another is this page at Stokely Consulting:

<http://www.stokely.com/unix.sysadm.resources/vrsctrl.html>

The `rc` Bourne shell script and associated documentation can be found at:

<http://www.cs.duke.edu/~des/toolman/>
<ftp://ftp.cs.duke.edu/pub/des/scripts/>

`rce` (RCs Edit), written by Rune Mossige <runemo@rl.telia.no>, is another Bourne shell script that is similar to `rc`. It works only with RCS and will attempt to provide an appropriate keyword header for the types of any files created. You can get it, too, at the Toolman Web page.

If you have a script that works with revision-control systems, please let me know about it and I'll add a link on the Toolman Web page.

Notes

[1] My use of revision control tends to fall within the scope of modifying system-configuration files and Web pages. I'm not involved with any large-scale code-development projects or such, which might involve more complex requirements.

[2] We'll defer the discussion of whether or not this is a good practice, that is, working on the file in place. It probably isn't.

[3] A similar "trick" can be done via the "editor" used by the `edquota` program; see a future Toolman article.

[4] This may also be the subject of a future article, if they don't lock me up first.