The RCS MINI−HOWTO

Table of Contents

The RCS MINI−HOWTO

Robert Kiesling

v1.4, 14 August 1997

This document covers basic installation and usage of RCS, the GNU Revision Control System, under Linux. It also covers the installation of the `diff(1)` and `diff3(1)` utilities, which are necessary for RCS to operate. This document may be reproduced freely, in whole or in part, provided that any usage of this document conforms to the general copyright notice of the HOWTO series of the Linux Documentation Project. See the file COPYRIGHT for details. Send all complaints, suggestions, errata, and any miscellany to kiesling@terracom.net, so I can keep this document as complete and up to date as possible.

8.RCS and `emacs(1)` Version Control.

1.Overview of RCS.

RCS, the revision control system, is a suite of programs that tracks changes in text files and controls shared access to files in work group situations. It is generally used to maintain source code modules. It lends itself to tracking revisions of document files as well.

RCS was written by Walter F. Tichy and Paul Eggert. The latest version which has been ported to Linux is RCS Version 5.7. There is also a semi−official, threaded version available. Much of the information in this HOWTO is taken from the RCS man pages.

RCS includes the `rcs(1)` program, which controls RCS archive file attributes, `ci(1)` and `co(1)`, which check files in and out of RCS archives, `ident(1)`, which searches RCS archives by keyword identifiers, `rcsclean(1)`, a program to clean up files that are not being worked on or haven't changed, `rcsdiff(1)`, which runs `diff(1)` to compare the revisions, `rcsmerge(1),` which merges two RCS branches into a single working file, and `rlog(1),` which prints RCS log messages.

Files archived by RCS may be text of any format, or binary if the `diff` program used to generate change files handles 8−bit data. Files may optionally include identification strings to aid in tracking by `ident(1).` RCS uses the utilities `diff(1)` and `diff3(3)` to generate the change files between revisions. A RCS archive consists of the initial revision of a file, which is version 1.1, and a series of change files, one for each revision. Each time a file is checked out of an archive with `co(1)`, edited, and checked back into the archive with `ci(1)`, the version number is increased, for example, to 1.2, 1.3, 1.4, and so on for successive revisions.

The archives themselves commonly reside in a `./RCS` subdirectory, although RCS has other options for archive storage.

For an overview of RCS, see the `rcsintro(1)` manual page.

2.System requirements.

RCS needs `diff(1)` and `diff3(3)` to generate the context diff files between revisions. The diff utilities suite needs to be installed on your system, and when you install RCS, the software will check for its presence.

Precompiled diffutils binaries are available at:

`ftp://sunsite.unc.edu/pub/Linux/utils/text/diffutils-2.6.bin.ELF.tar.gz`
and its mirror sites. If you need to compile `diff(1)`, et al., from source, it is located at:

`ftp://prep.ai.mit.edu/pub/gnu/diffutils-2.7.tar.gz`
and its mirror sites.

You will also need to have the ELF libraries installed on your system if you want to install pre−built binaries. See the ELF−HOWTO for further details.

3.Compiling RCS from Source.

Get the source distribution of RCS Version 5.7. It is available at

```
ftp://sunsite.unc.edu/pub/Linux/devel/vc/rcs-5.7.src.tar.gz
```

and its mirrors. After you have unpacked the archive into your source tree, you need to configure RCS for your system. This is done via the `configure` script in the source directory, which you need to execute first. This will generate a `Makefile` and the appropriate `conf.sh` for your system. You can then type

```
make install
```

which will build the binaries. At some point you may need to `su` to root so the binaries can be installed in the correct directories.

4.Creating and maintaining archives.

The program `rcs(1)` does the work or creating archives and modifying their attributes. A summary of `rcs(1)` options may be found in the `rcs(1)` manual page.

The easiest way to create an archive is first to `mkdir RCS` in the current directory, then initialize the archive with the

```
rcs -i name_of_work_file
```

command. This creates and archive with the name `./RCS/name_of_work_file,v` and requests a text message describing the archive, but it does not deposit any revisions in the archive. You can turn on or off strict archive locking with the commands

```
rcs -L name_of_work_file
```

and

```
rcs -U name_of_work_file
```

respectively. There are other options for controlling access to the archive, setting its format, and setting revision numbers, which are covered in the `rcs(1)` manual page.

5.ci(1) and co(1).

`ci(1)` and `co(1)` are the commands used to check files in and out of their RCS archives. The `ci(1)` command may also be used to a check a file both in and out of an archive. In their simplest forms, `ci(1)` and `co(1)` take only the name of the working file.

```
ci name_of_work_file
```
and

```
co name_of_work_file
```
The command form

```
ci −l name_of_work_file
```
checks in the file with locking enabled, and

```
co −l name_of_work_file
```
is performed automatically. That is, `ci −l` checks the file out again with locking enabled.

```
ci −u name_of_work_file
```
checks the file into the archive, and checks it out again with locking disabled. In all cases, the user is prompted for a log message.

`ci(1)` will also create a RCS archive if one does not exist already.

If you don't specify a revision, `ci(1)` increments the version number of the last revision locked in the archive, and appends the revised working file to it. If you specify a revision on an existing branch, it must be higher than the existing revision numbers. `ci(1)` will also create a new branch if you specify the revision of a branch which does not exist. See the `ci(1)` and `co(1)` man pages for details.

`ci(1)` and `co(1)` have various options for interactive and non−interactive use. Again, see the `ci(1)` and `co(1)` man pages for details.

6.Revision histories.

The `rlog(1)` program provides information about the archive file and the logs of each revision stored in it. A command like

```
rlog work_file_name
```
will print the version history of the file, each revision's creation date and `userids` of author and the person who locked the file. You can specify archive attributes and revision parameters to view.

7.Including RCS data in working files.

`co(1)` maintains a list of keywords of the RCS database which are expanded when the working file is checked out. The keyword `$Id$` in a document will expand to a string which contains the file name, revision number, the date checked out, the author, the revision status, and the locker, if any. Including the keyword `$Log$` will expand to the document's revision history log.

These and other keywords may be used as search criteria of the RCS archive. See the `ident(1)` man page for further details.

8.RCS and `emacs(1)` Version Control.

The Version Control facility of `emacs(1)` works as a front end to RCS. This information applies specifically to Version 19.34 of GNU Emacs, which is provided with the major Linux distributions. When editing a file with `emacs(1)` which is registered with RCS, the command `vc−toggle−read−only` (bound to `C−x C−q` by default) will check a file in to the emacs's Version Control, and then into RCS. Emacs will open a buffer where you can type a log message to be included in the RCS log. When you are finished typing a log entry, type `C−c C−c` to terminate your input and proceed with the check−in process.

If you have selected strict locking for the file with RCS, you must re−lock the file for editing by `emacs(1)`. You can check the file out for emacs's Version Control with the command `%` in buffer−menu mode.

For more information, see the GNU Emacs Manual and the Emacs info pages.
