A

History of RCS Releases

Walter Tichy started developing RCS when he was based at the Computer Science Department of Purdue University in the early 1980s. Since then, many different versions of RCS have been released by Purdue University. Several developers have been involved in the RCS project, and they have all added significant new functionality to RCS.

RCS versions are identified by a two part *Release.Revision* type version id number. This is similar in style to the RCS version id number, which RCS uses to identify the versions of a source file. The *Revision* number is incremented every time a new version of the RCS software is made available to the general public. However, the *Release* number is only incremented when significant new functionality is added. For this reason, two versions of RCS with the same *Release* number should have broadly similar functionality, (*e.g.*, version 4.3 has similar functionality to version 4.4, but version 5.5 has significantly different functionality).

If you received your RCS software on a tape, the RCS version number should be clearly identified on the tape. If you received your RCS software via electronic file transfer, the file name should identify which version of RCS you have got (*e.g.*, if the RCS source code is contained in a file named *rcs-5.5.tar.Z*, you have got version 5.5 of the RCS software). However, if you are unsure about the origin of the RCS software that you are currently using, you can always use the *ident* command, (which was described earlier,) to find out which release you are using. This command will retrieve the identification marker(s) which will identify the version number(s) of the original source file(s) used. For example the following command will find out which version of the source file *rcs.c* was used to compile the */usr/new/rcs* file (which is probably the *rcs* command you are currently using).

```
% ident /usr/new/rcs
```

It is important to remember that, the version number of the *rcs.c* source file will not necessarily correspond with the version number of RCS. This is because, when a new version of RCS is being created, some source files are not updated at all while other source files are updated several times. Hence, the version number of some source files can stay the same, while others may be incremented several times. Luckily, the RCS developers have adopted the convention that the release number of all RCS source files should correspond to the release number of the RCS software. This means that you know, if your RCS software was compiled using version 4.12 of the source file *rcs.c*, you must have release 4 of the RCS software. However, you cannot tell whether you have version 4.2 or version 4.3 of the RCS software.

It is important for you to know about the differences between each of the RCS releases because, even if you are currently using the most recent release of RCS, you might have to deal with compatibility problems caused by RCS files which were created by an earlier release of the RCS software. The following table summarizes the significant differences between the various releases of RCS.

Table A-1: Changes in RCS

Release	Year	Major Changes
Release 2	1982	RCS file names are generated by substituting ".v" in the place of the file suffix. This is significantly different from all other releases of RCS which append ",v" to the end of the file name. For example, a file named *sample.c* would have a corresponding RCS file named *sample.v* with release 2, but a RCS file named *sample.c,v* with any other release. The original suffix of the file is stored in the RCS file itself, because it would not be possible to guess it from the RCS file name. Later releases of RCS have the necessary code to read and understand release 2 RCS files. However, a special compile time option must be selected if this feature is to be enabled.
Release 3	1983	This release changed over to the ",v" type ending for RCS file names. Purdue University also released an enhanced *make* command which has built in rules for dealing with RCS files. (Previous releases of the *make* command had problems dealing with the ",v" file name ending.) This release of RCS was distributed with release 4.3 of BSD UNIX.
Release 4	1989	A new feature was added to allow users to set a default branch. This release was distributed through the Free Software Foundation (GNU). Major improvements were made to the RCS code to support portability.
Release 5	1990	The *co* and *ci* commands were enhanced to be able to deal with non-text files.

A-1

Release	Year	Major Changes
		Date handling was improved so that RCS will be able to handle dates beyond the end of the century. In addition, all check-in times are stored in GMT (Greenwich Mean Time) so that problems will not arise when sharing files across time zones. Options were added to allow users to select how keywords should be expanded. This release has features for enhanced compatibility with other releases of RCS. It will, whenever possible, write a file which is compatible with release 3/4 of RCS. The *co* command can be instructed to emulate an older version of RCS with the *-v* option flag. In addition, it has anticipated the need to read RCS files written by a future release of RCS; it ignores any fields it does not understand.

The recent versions of RCS are much more portable than the earlier versions. The original versions of RCS were only tested on BSD UNIX running on PDPs and Vaxen. However, RCS has now been ported to many different hardware and software platforms. As a result, RCS should now work satisfactorily on practically any UNIX type system. If you encounter any compatibility or other problems with the RCS software running on your system, you should send a mail message to the mailing list *rcs-bugs@cs.purdue.edu* and someone will help you out.

B

Converting SCCS files into RCS files

If you are an existing user of SCCS, you may be reluctant to change over to using RCS, because of all of the existing SCCS files which you must still maintain. Luckily for you, there exists a utility for automatically converting existing SCCS files into RCS files. This utility is called *sccstorcs*.

For example you could use the following command to convert the SCCS file *s.sample.c* into a corresponding RCS file *sample.c,v*.

```
% sccstorcs s.sample.c
```

The *sccstorcs* command can also be used for converting a large number of SCCS files at once. For example the following command converts all of the SCCS files in the current directory into RCS files.

```
% sccstorcs s.*
```

The really valuable thing about *sccstorcs* is that, it preserves all of the versions stored in the SCCS file. Thus the newly created RCS file *sample.c,v* would have exactly the same versions as *s.sample.c*. The versions of *sample.c,v* would also have the same attributes as the versions of *s.sample.c* including their creation date, commentary and author. The *sccstorcs* command also maintains the descriptive text and the access control list of the original SCCS file if it had any.

Unfortunately it is not possible to preserve all of the information from the SCCS file when transforming it to a RCS file. RCS has no concept of *removed deltas*, *MR numbers* or *cutoff points*. As a result if your SCCS files contained any of these features they will be lost in the RCS file.

The *sccstorcs* command works by retrieving each version of the SCCS file into the working directory and then checking it into the corresponding RCS file with the *ci* command. *sccstorcs uses the -w, r, -m* and *-d* options to ensure that the versions in the newly created RCS file have the correct author, version number, commentary and creation date. If you use the *-v* (verbose), option flag with the *sccstorcs* command it will issue messages informing you exactly which *ci* commands it is issuing.

If you are of a nervous disposition, you can use the **sccstorcs -t** option to trace the *ci* commands that would be used by *sccstorcs* before you run it for real. For example the following command will inform you which *ci* commands would be used by *sccstorcs* to convert the SCCS file *s.sample.c* into a RCS file.

```
% sccstorcs -t s.sample.c
```

However, there is no need for such nervousness, *sccstorcs* is very conservative and will abort rather than overwrite an existing RCS file or working file.

The *sccstorcs* utility was written by Kenneth Greer. It is sometimes (but not always) distributed with RCS itself.

C

Storing Differences

This appendix will contain a comparison of reverse deltas and interleaved deltas (a point of fascination for many people).

D

Date Formats

RCS allows you to specify dates in *free format* which means that the system is very flexible about the format in which you specify dates. If a RCS command requires you to specify a date, RCS will parse the date/time string you provide into an internal form that it will use. The date/time parsing routines used by RCS are general purpose routines written by Ken Harrenstein, these general purpose routines are probably more powerful than you normally require because they allow you to specify dates exactly to the second. In general RCS will parse date strings in the way that you would intuitively expect. However we have included this appendix describing how dates are parsed by RCS so that you can learn exactly what is acceptable and what is not.

The following list is a sample of the type of date formats that RCS is able to parse.

```
10-July-1989 13:02:25-GMT
07/10/89 13:02:25-GMT
gmt, 1989 1:02:25pm jul 10
Monday,July 10 13:02:25 GMT 1989
```

All of the above date strings would be interpreted by RCS to mean 2 minutes and 25 seconds after 1pm Greenwich Mean Time on Monday the 10th of July 1989. Internally RCS stores dates in a format *YY.MM.DD.hh.mm.ss*; where *YY* is the number of years past 1900, *MM* is the month number, *DD* is the day number within the month, *hh* is the hour number in 24 hour format and *ss* is the number of seconds past the minute. For the example date given above, that would be "89.07.10.13.02.25" . You will notice that this format follows the most significant bit first convention. This format is used because dates can be easily compared by comparing the ASCII values of the RCS strings used to represent them.

RCS allows you to specify (in order of significance) the year, month, day, hour, minute and second. However, very few people specify values for all of these parameters, hence RCS has two simple rules which it uses for defaulting parameters which are not specified.

- Any parameter which is more significant than those you provide is defaulted to the current value. For example if you specify "10pm", RCS will interpret this to mean 10pm today; if you specify "10-September", RCS will interpret this to mean the 10th of September this year.
- Any parameter which is less significant than those you provide is defaulted to the minimum value. For example if you specify "10am", RCS will interpret this to mean 10:00:00; if you specify "1990", RCS will interpret this to mean 0:00:00 on the 1st of January 1990.

If a date is specified as a series of numbers separated by either the '/' or '-' characters, the american date notation is assumed. (i.e. a date in the form of xx/xx/xx is interpreted as month/day/year. and a date in the form of xx/xx is interpreted as month/day.) For example 10/12/90 is interpreted as the 12th of October 1990 rather than the 10th of December 1990 as it would be interpreted in Europe.

RCS will recognize the ascii strings representing the english names of the months of the year.† It is also possible to use any unambiguous abbreviation of the months of two letters or more. For example "ja" is an acceptable abbreviation for January, but "ju" is not an acceptable abbreviation for July because it can be confused with June. If you are using ascii month names you can give the day of the month and the month name in any order because they will not be confused. For example "10-dec-90" and "dec-10-90" will both be interpreted as the 10th of December 1990.

RCS will recognize the ascii strings for the days of the week (in english) or any abbreviation of these of two letters or more. Specifying a day of the week on its own is not considered a valid date specification, however if you include a day of the week along with a date, the day of the week must be correct for the date given. Hence, specifying a day of the week is only useful as a check on the correctness of the date specified. For example, the string "Sat 1-Sept-1990" will be understood to mean the first of September 1990 (which happens to be a saturday). However the string "Sat" or the string "Sat 2-Sept-1990" will be rejected.

If you specify a date/time string consisting of a single number, RCS will assume that this number is a day of the month. For example the date/time string "10" will be parsed to 0:00:00 on the 10th of this month. However, if the single number is followed by either "am" or "pm", RCS will treat it as an hour number. For example the date/time string "10pm" will be parsed as 22:00:00 today.

If you specify a time to RCS, you must use the HH:MM:SS notation. For example the string "7:06:05" is interpreted as six minutes and 5 seconds past seven o'clock. RCS understands the ascii strings "am", "pm", "noon" and "midnight" or any two or more letter abbreviation of them. Unless you specify either am or pm the system assumes you are using a 24 hour clock. You can omit either the minutes or the seconds and they will be defaulted to zero. If you specify either the number of minutes or the number of seconds you must give exactly two digits for each (e.g. "7:6:5" is not a valid time string while "7:06:05" is).

RCS normally interprets times as referring to the time zone that is currently active in your computer (i.e. local time). However, if you want the time you specify to be interpreted as a time in a different time zone you can explicitly state the time zone that you are using. RCS will recognize the time zone abbreviations given in the following table.

† The RCS date parsing routines are not case sensitive (i.e. uppercase and lowercase characters can be interchanged in ascii strings).

Table D-1: Timezone Abbreviations

Abbreviation	Meaning	Delay Relative to GMT
gmt	Greenwich Mean Time	0
gst	Greenwich Standard Time	0
ast	Atlantic Standard Time	4 hours
est	Eastern Standard Time	5 hours
cst	Central Standard Time	6 hours
mst	Mountain Standard Time	7 hours
pst	Pacific Standard Time	8 hours
yst	Yukon Standard Time	9 hours
hst	Hawaii Standard Time	10 hours
bst	Bering Standard Time	11 hours
gdt	Greenwich Daylight Saving Time	-1 hour
adt	Atlantic Daylight Saving Time	3 hours
edt	Eastern Daylight Saving Time	4 hours
cdt	Central Daylight Saving Time	5 hours
mdt	Mountain Daylight Saving Time	6 hours
pdt	Pacific Daylight Saving Time	7 hours
ydt	Yukon Daylight Saving Time	8 hours
hdt	Hawaii Daylight Saving Time	9 hours
bdt	Bering Daylight Saving Time	10 hours
daylight	Local Daylight Saving Time	??
std	Local Standard Time	??
standard	Local Standard Time	??

For example if you were working in Boston during the winter, your computer would be working on Eastern Daylight Time (EDT). You could retrieve the latest version of the file *sample.c* created since 10am this morning (local time) by issuing the command :

```
% co "-d10:00" sample.c
```

However, since "14:00 GMT" is equivalent to "10:00 EDT", you could also have used the command :

```
% co "-d14:00 GMT" sample.c
```

You will probably not use the time zone feature very frequently, because it is unlikely that you would want to specify a time zone unless you were logged in over a wide are network and were confused about what time zone the computer was operating in.