RCS-Docs

Walter F. al and AmigaGuide translation JBHR

COLLABORATORS

	TITLE : RCS-Docs	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Walter F. al and AmigaGuide translation JBHR	September 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

RCS-Docs

1.1 Revision Control System

```
                RCS 5.6.0.1 - Revision Control System 5.6.0.1 for the Amiga
*********************************************************


            RCSINTRO
             introduction to RCS commands

Commands:


            CI
             check in RCS revisions

            CO
             check out RCS revisions

            IDENT
             identify files

            MERGE
             three-way file merge

            RCS
             change RCS file attributes

            RCSCLEAN
             clean up working files

            RCSDIFF
             compare RCS revisions

            RCSFREEZE
             freeze a configuration of sources checked in under RCS

            RCSMERGE
             merge RCS revisions
```

RLOG
print log messages and other information about RCS files

File format:

RCSFILE
format of RCS file

1.2 RCSINTRO

NAME
rcsintro - introduction to RCS commands

DESCRIPTION
The Revision Control System (RCS) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, and form letters.

The basic user interface is extremely simple. The novice only needs to learn two commands:
ci
and
co
. ci, short for "check in", deposits the contents of a file into an archival file called an RCS file. An RCS file contains all revisions of a particular file. co, short for "check out", retrieves revisions from an RCS file.

Functions of RCS

- Store and retrieve multiple revisions of text. RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.

- Maintain a complete history of changes. RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of check-in, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.

- Resolve access conflicts. When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.

- Maintain a tree of revisions. RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.

- Merge revisions and resolve conflicts. Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged affect the same sections of code, RCS alerts the user about the overlapping changes.

- Control releases and configurations. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.

- Automatically identify each revision with name, revision number, creation time, author, etc. The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.

- Minimize secondary storage. RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting Started with RCS

Suppose you have a file f.c that you wish to put under control of RCS. If you have not already done so, make an RCS directory with the command

```
mkdir RCS
```

Then invoke the check-in command

```
ci f.c
```

This command creates an RCS file in the RCS directory, stores f.c into it as revision 1.1, and deletes f.c. It also asks you for a description. The description should be a synopsis of the contents of the file. All later check-in commands will ask you for a log entry, which should summarize the changes that you made.

Files in the RCS directory are called RCS files; the others are called working files. To get back the working file f.c in the previous example, use the check-out command

```
co f.c
```

This command extracts the latest revision from the RCS file and writes it into f.c. If you want to edit f.c, you must lock it as you check it out with the command

```
co  -l  f.c
```

You can now edit f.c.

Suppose after some editing you want to know what changes that you have made. The command

```
rcsdiff  f.c
```

tells you the difference between the most recently checked-in version and the working file. You can check the file back in by invoking

```
ci  f.c
```

This increments the revision number properly.

If ci complains with the message

```
ci error: no lock set by your name
```

then you have tried to check in a file even though you did not lock it when you checked it out. Of course, it is too late now to do the check-out with locking, because another check-out would overwrite your modifications. Instead, invoke

```
rcs  -l  f.c
```

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. All that locking prevents is a check-in by anybody but the locker.

If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for check-in; all others still do. Turning strict locking off and on is done with the commands

```
rcs  -U  f.c     and     rcs  -L  f.c
```

If you don't want to clutter your working directory with RCS files, create a subdirectory called RCS in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. (Actually, pairs of RCS and working files can be specified in three ways: (a) both are given, (b) only the working file is given, (c) only the RCS file is given. Both RCS and working files may have arbitrary path prefixes;

RCS commands pair them up intelligently.)

To avoid the deletion of the working file during check-in (in case you want to continue editing or compiling), invoke

```
ci -l f.c     or     ci -u f.c
```

These commands check in f.c as usual, but perform an implicit check-out. The first form also locks the checked in revision, the second one doesn't. Thus, these options save you one check-out operation. The first form is useful if you want to continue editing, the second one if you just want to read the file. Both update the identification markers in your working file (see below).

You can give ci the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2. The command

```
ci -r2 f.c     or     ci -r2.1 f.c
```

assigns the number 2.1 to the new revision. From then on, ci will number the subsequent revisions with 2.2, 2.3, etc. The corresponding co commands

```
co -r2 f.c     and     co -r2.1 f.c
```

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. co without a revision number selects the latest revision on the trunk, i.e. the highest revision with a number consisting of two fields. Numbers with more than two fields are needed for branches. For example, to start a branch at revision 1.3, invoke

```
ci -r1.3.1 f.c
```

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see

```
rcsfile
.
```

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker

```
$Id$
```

into your text, for instance inside a comment. RCS will replace this marker with a string of the form

```
$Id: filename revision date time author state $
```

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps

```
     the markers up to date automatically.  To propagate the
     markers into your object code, simply put them into literal
     character strings.  In C, this is done as follows:

          static char rcsid[] = "$Id$";

     The command ident extracts such markers from any file, even
     object code and dumps.  Thus, ident lets you find out which
     revisions of which modules were used in a given program.

     You may also find it useful to put the marker $Log$ into
     your text, inside a comment.  This marker accumulates the
     log messages that are requested during check-in.  Thus, you
     can maintain the complete history of your file directly
     inside it.  There are several additional identification
     markers; see
                co
                 for details.

IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.1; Release Date: 1991/04/21.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                ,
                ident
                ,
                rcs
                ,
                rcsdiff
                ,
                rcsintro
                ,

                rcsmerge
                ,
                rlog
                     Walter F. Tichy, RCS--A System for Version Control,
     Software--Practice & Experience 15, 7 (July 1985), 637-654.
```

1.3 CI

```
                NAME
     ci - check in RCS revisions

SYNOPSIS
     ci [options] file ...

DESCRIPTION
```

ci stores new revisions into RCS files. Each pathname matching an RCS suffix is taken to be an RCS file. All others are assumed to be working files containing new revisions. ci deposits the contents of each working file into the corresponding RCS file. If only a working file is given, ci tries to find the corresponding RCS file in an RCS subdirectory and then in the working file's directory. For more details, see FILE NAMING below.

For ci to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file if non-strict locking is used (see

rcs

). A lock held by someone else may be broken with the rcs command.

Unless the -f option is given, ci checks whether the revision to be deposited differs from the preceding one. If not, instead of creating a new revision ci reverts to the preceding one. To revert, ordinary ci removes the working file and any lock; ci -l keeps and ci -u removes any lock, and then they both generate a new working file much as if co -l or co -u had been applied to the preceding revision. When reverting, any -n and -s options apply to the preceding revision.

For each revision deposited, ci prompts for a log message. The log message should summarize the change and must be terminated by end-of-file or by a line containing . by itself. If several files are checked in ci asks whether to reuse the previous log message. If the standard input is not a terminal, ci suppresses the prompt and uses the same log message for all files. See also -m.

If the RCS file does not exist, ci creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, ci requests descriptive text (see -t below).

The number rev of the deposited revision can be given by any of the options -f, -I, -k, -l, -M, -q, -r, or -u. rev may be symbolic, numeric, or mixed. If rev is $, ci determines the revision number from keyword values in the working file.

If rev is a revision number, it must be higher than the latest one on the branch to which rev belongs, or must start a new branch.

If rev is a branch rather than a revision number, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that

branch. If rev indicates a non-existing branch, that branch is created with the initial revision numbered rev.1.

If rev is omitted, ci tries to derive the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1.

If rev is omitted and the caller has no lock, but owns the file and locking is not set to strict, then the revision is appended to the default branch (normally the trunk; see the -b option of
rcs
).

Exception: On the trunk, revisions can be appended to the end, but not inserted.

OPTIONS

-r[rev]
checks in a revision, releases the corresponding lock, and removes the working file. This is the default.

The -r option has an unusual meaning in ci. In other RCS commands, -r merely specifies a revision number, but in ci it also releases a lock and removes the working file. See -u for a tricky example.

-l[rev]
works like -r, except it performs an additional co -l for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

-u[rev]
works like -l, except that the deposited revision is not locked. This lets one read the working file immediately after checkin.

The -l, -r, and -u options are mutually exclusive and silently override each other. For example, ci -u -r is equivalent to ci -r because -r overrides -u.

-f[rev]
forces a deposit; the new revision is deposited even it is not different from the preceding one.

-k[rev]
searches the working file for keyword values to determine its revision number, creation date, state, and author (see

co
), and assigns these values to the deposited revision, rather than computing them locally. It also generates a default login message noting the login of the caller and the actual checkin date. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the -k option at these sites to preserve the original number, date, author, and state. The extracted keyword values and the default log message may be overridden with the options -d, -m, -s, -w, and any option that carries a revision number.

-q[rev]
quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless -f is given.

-I[rev]
interactive mode; the user is prompted and questioned even if the standard input is not a terminal.

-d[date]
uses date for the checkin date and time. The date is specified in free format as explained in
co
. This is useful for lying about the checkin date, and for -k if no date is available. If date is empty, the working file's time of last modification is used.

-M[rev]
Set the modification time on any new working file to be the date of the retrieved revision. For example, ci -d -M -u f does not alter f's modification time, even if f's contents change due to keyword substitution. Use this option with care; it can confuse make.

-mmsg
uses the string msg as the log message for all revisions checked in.

-nname
assigns the symbolic name name to the number of the checked-in revision. ci prints an error message if name is already assigned to another number.

-Nname
same as -n, except that it overrides a previous assignment of name.

-sstate
sets the state of the checked-in revision to the identifier state. The default state is Exp.

-tfile

writes descriptive text from the contents of the named file into the RCS file, deleting the existing text. The file may not begin with -.

-t-string
Write descriptive text from the string into the RCS file, deleting the existing text.

The -t option, in both its forms, has effect only during an initial checkin; it is silently ignored otherwise.

During the initial checkin, if -t is not given, ci obtains the text from standard input, terminated by end-of-file or by a line containing . by itself. The user is prompted for the text if interaction is possible; see -I.

For backward compatibility with older versions of RCS, a bare -t option is ignored.

-wlogin
uses login for the author field of the deposited revision. Useful for lying about the author, and for -k if no author is available.

-Vn Emulate RCS version n. See
co
for details.

-xsuffixes
specifies the suffixes for RCS files. A nonempty suffix matches any pathname ending in the suffix. An empty suffix matches any pathname of the form RCS/file or path/RCS/file. The -x option can specify a list of suffixes separated by /. For example, -x,v/ specifies two suffixes: ,v and the empty suffix. If two or more suffixes are specified, they are tried in order when looking for an RCS file; the first one that works is used for that file. If no RCS file is found but an RCS file can be created, the suffixes are tried in order to determine the new RCS file's name. The default for suffixes is installation-dependent; normally it is ,v/ for hosts like Unix that permit commas in file names, and is empty (i.e. just the empty suffix) for other hosts.

FILE NAMING
Pairs of RCS files and working files may be specified in three ways (see also the example section).

1) Both the RCS file and the working file are given. The RCS pathname is of the form path1/workfileX and the working pathname is of the form path2/workfile where path1/ and path2/ are (possibly different or empty) paths, workfile is a filename, and X is an RCS suffix. If X is empty, path1/ must be RCS/ or must end in /RCS/.

2) Only the RCS file is given. Then the working file is created in the current directory and its name is derived from the name of the RCS file by removing path1/ and the suffix X.

3) Only the working file is given. Then ci considers each RCS suffix X in turn, looking for an RCS file of the form path2/RCS/workfileX or (if the former is not found and X is nonempty) path2/workfileX.

If the RCS file is specified without a path in 1) and 2), ci looks for the RCS file first in the directory ./RCS and then in the current directory.

ci reports an error if an attempt to open an RCS file fails for an unusual reason, even if the RCS file's pathname is just one of several possibilities. For example, to suppress use of RCS commands in a directory d, create a regular file named d/RCS so that casual attempts to use RCS commands in d fail because d/RCS is not a directory.

EXAMPLES

Suppose ,v is an RCS suffix and the current directory contains a subdirectory RCS with an RCS file io.c,v. Then each of the following commands check in a copy of io.c into RCS/io.c,v as the latest revision, removing io.c.

```
ci io.c;    ci RCS/io.c,v;   ci io.c,v;
ci io.c RCS/io.c,v;    ci io.c io.c,v;
ci RCS/io.c,v io.c;    ci io.c,v io.c;
```

Suppose instead that the empty suffix is an RCS suffix and the current directory contains a subdirectory RCS with an RCS file io.c. The each of the following commands checks in a new revision.

FILE MODES

An RCS file created by ci inherits the read and execute permissions from the working file. If the RCS file exists already, ci preserves its read and execute permissions. ci always turns off all write permissions of RCS files.

FILES

Several temporary files may be created in the directory containing the working file, and also in the temporary directory (see TMPDIR under ENVIRONMENT). A semaphore file or files are created in the directory containing the RCS file. With a nonempty suffix, the semaphore names begin with the first character of the suffix; therefore, do not specify an suffix whose first character could be that of a working filename. With an empty suffix, the semaphore names end with _ so working filenames should not end in _.

ci never changes an RCS or working file. Normally, ci unlinks the file and creates a new one; but instead of breaking a chain of one or more symbolic links to an RCS

file, it unlinks the destination file instead. Therefore, ci breaks any hard or symbolic links to any working file it changes; and hard links to RCS files are ineffective, but symbolic links to RCS files are preserved.

The effective user must be able to search and write the directory containing the RCS file. Normally, the real user must be able to read the RCS and working files and to search and write the directory containing the working file; however, some older hosts cannot easily switch between real and effective users, so on these hosts the effective user is used for all accesses. The effective user is the same as the real user unless your copies of ci and co have setuid privileges. As described in the next section, these privileges yield extra security if the effective user owns all RCS files and directories, and if only the effective user can write RCS directories.

Users can control access to RCS files by setting the permissions of the directory containing the files; only users with write access to the directory can use RCS commands to change its RCS files. For example, in hosts that allow a user to belong to several groups, one can make a group's RCS directories writable to that group only. This approach suffices for informal projects, but it means that any group member can arbitrarily change the group's RCS files, and can even remove them entirely. Hence more formal projects sometimes distinguish between an RCS administrator, who can change the RCS files at will, and other project members, who can check in new revisions but cannot otherwise change the RCS files.

ENVIRONMENT

RCSINIT

options prepended to the argument list, separated by spaces. A backslash escapes spaces within an option. The RCSINIT options are prepended to the argument lists of most RCS commands. Useful RCSINIT options include -q, -V, and -x.

TMPDIR

Name of the temporary directory. If not set, the environment variables TMP and TEMP are inspected instead and the first value found is taken; if none of them are set, a host-dependent default is used, typically /tmp.

DIAGNOSTICS

For each revision, ci prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 5.9; Release Date: 1991/10/07.

SEE ALSO

co
,
ident
, make,
rcs
,
rcsclean
,
rcsdiff
,

rcsintro
,
rcsmerge
,
rlog
,
rcsfile
Walter F. Tichy, RCS--A System for Version Control, Software--Practice & Experience 15, 7 (July 1985), 637-654.

1.4 CO

NAME
co - check out RCS revisions

SYNOPSIS
co [options] file ...

DESCRIPTION
co retrieves a revision from each RCS file and stores it into the corresponding working file.

Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in

ci
.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. Checkout with locking fails if the revision to be checked out is currently locked by another user. (A lock may be broken with

rcs
.) Checkout with locking also requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is

empty. Checkout without locking is not subject to access-list restrictions, and is not affected by the presence of locks.

A revision is selected by options for revision or branch number, checkin date/time, author, or state. When the selection options are applied in combination, co retrieves the latest revision that satisfies all of them. If none of the selection options is specified, co retrieves the latest revision on the default branch (normally the trunk, see the -b option of
rcs
). A revision or branch number may be attached to any of the options -f, -I, -l, -M, -p, -q, -r, or -u. The options -d (date), -s (state), and -w (author) retrieve from a single branch, the selected branch, which is either specified by one of -f, ..., -u, or the default branch.

A co command applied to an RCS file with no revisions creates a zero-length working file. co always performs keyword substitution (see below).

OPTIONS

-r[rev]
retrieves the latest revision whose number is less than or equal to rev. If rev indicates a branch rather than a revision, the latest revision on that branch is retrieved. If rev is omitted, the latest revision on the default branch (see the -b option of
rcs
) is
retrieved. If rev is $, co determines the revision number from keyword values in the working file. Otherwise, a revision is composed of one or more numeric or symbolic fields separated by periods. The numeric equivalent of a symbolic field is specified with the -n option of the commands
ci
and
rcs
.

-l[rev]
same as -r, except that it also locks the retrieved revision for the caller.

-u[rev]
same as -r, except that it unlocks the retrieved revision if it was locked by the caller. If rev is omitted, -u retrieves the revision locked by the caller, if there is one; otherwise, it retrieves the latest revision on the default branch.

-f[rev]
forces the overwriting of the working file; useful in connection with -q. See also FILE MODES below.

-kkv Generate keyword strings using the default form, e.g. $Revision: 5.7 $ for the Revision keyword. A locker's name is inserted in the value of the Header, Id, and Locker keyword strings only as a file is being locked, i.e. by ci -l and co -l. This is the default.

-kkvl
Like -kkv, except that a locker's name is always inserted if the given revision is currently locked.

-kk Generate only keyword names in keyword strings; omit their values. See KEYWORD SUBSTITUTION below. For example, for the Revision keyword, generate the string $Revision$ instead of $Revision: 5.7 $. This option is useful to ignore differences due to keyword substitution when comparing different revisions of a file.

-ko Generate the old keyword string, present in the working file just before it was checked in. For example, for the Revision keyword, generate the string $Revision: 1.1 $ instead of $Revision: 5.7 $ if that is how the string appeared when the file was checked in. This can be useful for binary file formats that cannot tolerate any changes to substrings that happen to take the form of keyword strings.

-kv Generate only keyword values for keyword strings. For example, for the Revision keyword, generate the string 5.7 instead of $Revision: 5.7 $. This can help generate files in programming languages where it is hard to strip keyword delimiters like $Revision: $ from a string. However, further keyword substitution cannot be performed once the keyword names are removed, so this option should be used with care. Because of this danger of losing keywords, this option cannot be combined with -l, and the owner write permission of the working file is turned off; to edit the file later, check it out again without -kv.

-p[rev]
prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when co is part of a pipe.

-q[rev]
quiet mode; diagnostics are not printed.

-I[rev]
interactive mode; the user is prompted and questioned even if the standard input is not a terminal.

-ddate
retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to date. The date and time may be given in free format. The time zone LT stands for local time; other common time

zone names are understood. For example, the following dates are equivalent if local time is January 11, 1990, 8pm Pacific Standard Time, eight hours west of Coordinated Universal Time (UTC):

```
  8:00 pm lt
  4:00 AM, Jan. 12, 1990        note: default is UTC
  1990/01/12 04:00:00           RCS date format
  Thu Jan 11 20:00:00 1990 LT   output of ctime + LT
  Thu Jan 11 20:00:00 PST 1990  output of date
  Fri Jan 12 04:00:00 GMT 1990
  Thu, 11 Jan 1990 20:00:00 -0800
  Fri-JST, 1990, 1pm Jan 12
  12-January-1990, 04:00-WET
```

Most fields in the date and time may be defaulted. The default time zone is UTC. The other defaults are determined in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the time zone's current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date 20, 10:30 defaults to 10:30:00 UTC of the 20th of the UTC time zone's current month and year. The date/time must be quoted if it contains spaces.

-M[rev]
: Set the modification time on the new working file to be the date of the retrieved revision. Use this option with care; it can confuse make.

-sstate
: retrieves the latest revision on the selected branch whose state is set to state.

-w[login]
: retrieves the latest revision on the selected branch which was checked in by the user with login name login. If the argument login is omitted, the caller's login is assumed.

-jjoinlist
: generates a new revision which is the join of the revisions on joinlist. This option is largely obsoleted by

  rcsmerge
  but is retained for backwards compatibility.

  The joinlist is a comma-separated list of pairs of the form rev2:rev3, where rev2 and rev3 are (symbolic or numeric) revision numbers. For the initial such pair, rev1 denotes the revision selected by the above options -f, ..., -w. For all other pairs, rev1 denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, co joins revisions rev1 and rev3 with respect to rev2. This means that all changes that transform rev2 into rev1 are applied to a copy of rev3. This is particularly useful if rev1 and rev3 are the ends of two branches that have rev2 as a common ancestor. If rev1<rev2<rev3 on the same branch, joining generates a new revision which is like rev3, but with all changes that lead from rev1 to rev2 undone. If changes from rev2 to rev1 overlap with changes from rev2 to rev3, co reports overlaps as described in

merge
.

For the initial pair, rev2 may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. The options -l and -u lock or unlock rev1.

-Vn Emulate RCS version n, where n may be 3, 4, or 5. This may be useful when interchanging RCS files with others who are running older versions of RCS. To see which version of RCS your correspondents are running, have them invoke rlog on an RCS file; if none of the first few lines of output contain the string branch: it is version 3; if the dates' years have just two digits, it is version 4; otherwise, it is version 5. An RCS file generated while emulating version 3 will lose its default branch. An RCS revision generated while emulating version 4 or earlier will have a timestamp that is off by up to 13 hours. A revision extracted while emulating version 4 or earlier will contain dates of the form yy/mm/dd instead of yyyy/mm/dd and may also contain different white space in the substitution for Log.

-xsuffixes
Use suffixes to characterize RCS files. See
ci
for de-
tails.

KEYWORD SUBSTITUTION

Strings of the form $keyword$ and $keyword:...$ embedded in the text are replaced with strings of the form $keyword:value$ where keyword and value are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form $keyword$. On checkout, co replaces these strings with strings of the form $keyword:value$. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout. This automatic sub-

stitution can be modified by the -k options.

Keywords and their corresponding values:

$Author$
The login name of the user who checked in the revision.

$Date$
The date and time (UTC) the revision was checked in.

$Header$
A standard header containing the full pathname of the RCS file, the revision number, the date (UTC), the author, the state, and the locker (if locked).

Id Same as $Header$, except that the RCS filename is without a path.

$Locker$
The login name of the user who locked the revision (empty if not locked).

Log
The log message supplied during checkin, preceded by a header containing the RCS filename, the revision number, the author, and the date (UTC). Existing log messages are not replaced. Instead, the new log message is inserted after $Log:...$. This is useful for accumulating a complete change log in a source file.

$RCSfile$
The name of the RCS file without a path.

$Revision$
The revision number assigned to the revision.

$Source$
The full pathname of the RCS file.

$State$
The state assigned to the revision with the -s option of
rcs
or
ci
.

FILE MODES
The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless -kv is set or the file is checked out unlocked and locking is set to strict (see
rcs
).

If a file with the name of the working file exists already and has write permission, co aborts the checkout, asking

beforehand if possible. If the existing working file is not writable or -f is given, the working file is deleted without asking.

FILES

co accesses files much as

ci

does, except that it does not need to read the working file.

ENVIRONMENT

RCSINIT

options prepended to the argument list, separated by spaces. See

ci

for details.

DIAGNOSTICS

The RCS pathname, the working pathname, and the revision number retrieved are written to the diagnostic output. The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 5.7; Release Date: 1991/08/19.

SEE ALSO

ci
, ctime,
ident
, make,
rcs
,

rcsdiff
,
rcsintro
,
rcsmerge
,
rlog
,
rcsfile

Walter F. Tichy, RCS--A System for Version Control, Software--Practice & Experience 15, 7 (July 1985), 637-654.

LIMITS

Links to the RCS and working files are not preserved.

There is no way to selectively suppress the expansion of keywords, except by writing them differently. In nroff and troff, this is done by embedding the null-character \& into the keyword.

```
BUGS
     The -d option sometimes gets confused, and accepts  no  date
     before 1970.
```

1.5 IDENT

```
               NAME
     ident - identify files

SYNOPSIS
     ident [ -q ] [ file ... ]

DESCRIPTION
     ident searches for all  occurrences  of  the  pattern  $key-
     word:...$  in  the  named files or, if no file name appears,
     the standard input.

     These patterns are normally inserted  automatically  by  the
     RCS command
               co
               , but can also be inserted manually. The opt-
     ion -q suppresses the warning given if there are no patterns
     in a file.

     ident works on text files as well as object files and dumps.
     For example, if the C program in f.c contains

          char rcsid[] =  "$Id:  f.c,v  5.0  1990/08/22  09:09:36
          eggert Exp $";

     and f.c is compiled into f.o, then the command

          ident  f.c  f.o

     will output

          f.c:
              $Id: f.c,v 5.0 1990/08/22 09:09:36 eggert Exp $
          f.o:
              $Id: f.c,v 5.0 1990/08/22 09:09:36 eggert Exp $

IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.0; Release Date: 1990/08/22.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990 by Paul Eggert.

SEE ALSO

               ci
               ,
               co
               ,
               rcs
```

```
               ,
               rcsdiff
               ,
               rcsintro
               ,

               rcsmerge
               ,
               rlog
               ,
               rcsfile
                    Walter  F.  Tichy,  RCS--A  System  for  Version  Control,
     Software--Practice & Experience 15, 7 (July 1985), 637-654.
```

1.6 MERGE

```
               NAME
     merge - three-way file merge

SYNOPSIS
     merge [ -L label1 [ -L label3 ] ] [ -p ] [ -q ] file1  file2
     file3

DESCRIPTION
     merge incorporates all changes that lead from file2 to file3
     into  file1.   The  result  goes to standard output if -p is
     present, into file1 otherwise.  merge is useful for  combin-
     ing  separate  changes to an original.  Suppose file2 is the
     original, and both file1  and  file3  are  modifications  of
     file2.  Then merge combines both changes.

     An overlap occurs if both file1 and file3 have changes in  a
     common  segment  of lines.  On a few older hosts where diff3
     does not support the -E option, merge does not detect  over-
     laps,  and  merely supplies the changed lines from file3. On
     most hosts, if  overlaps  occur,  merge  outputs  a  message
     (unless  the -q option is given), and includes both alterna-
     tives in the result.  The alternatives are delimited as fol-
     lows:

          <<<<<<< file1
          lines in file1
          =======
          lines in file3
          >>>>>>> file3

     If there are overlaps, the user should edit the  result  and
     delete  one  of  the  alternatives.   If  the  -L label1 and
     -L label3 options are given, the labels are output in  place
     of the names file1 and file3 in overlap reports.

DIAGNOSTICS
     Exit status is 0 for no overlaps, 1 for some overlaps, 2 for
     trouble.
```

```
IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.3; Release Date: 1991/02/28.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO
     diff3, diff,
                rcsmerge
                ,
                co
                .
```

1.7 RCS

```
                NAME
     rcs - change RCS file attributes

SYNOPSIS
     rcs [ options ] file ...

DESCRIPTION
     rcs creates new RCS files or changes attributes of  existing
     ones.   An  RCS file contains multiple revisions of text, an
     access list, a change log, descriptive text, and  some  con-
     trol  attributes.   For rcs to work, the caller's login name
     must be on the access list, except if  the  access  list  is
     empty, the caller is the owner of the file or the superuser,
     or the -i option is present.

     Pathnames matching an RCS suffix denote RCS files; all  oth-
     ers  denote working files.  Names are paired as explained in

                ci
                .  Revision numbers use the syntax described in
                ci
                .

OPTIONS
     -i   Create and initialize a new RCS file, but do not  depo-
          sit  any revision.  If the RCS file has no path prefix,
          try to place it first into the subdirectory ./RCS,  and
          then  into  the  current  directory.   If  the RCS file
          already exists, print an error message.

     -alogins
          Append the login names appearing in the comma-separated
          list logins to the access list of the RCS file.

     -Aoldfile
          Append the access list of oldfile to the access list of
          the RCS file.

     -e[logins]
          Erase the login names appearing in the  comma-separated
```

list logins from the access list of the RCS file. If logins is omitted, erase the entire access list.

-b[rev]
Set the default branch to rev. If rev is omitted, the default branch is reset to the (dynamically) highest branch on the trunk.

-cstring
sets the comment leader to string. The comment leader is printed before every log message line generated by the keyword Log during checkout (see
co
). This is
useful for programming languages without multi-line comments. An initial ci , or an rcs -i without -c, guesses the comment leader from the suffix of the working file.

-ksubst
Set the default keyword substitution to subst. The effect of keyword substitution is described in
co
.
Giving an explicit -k option to co, rcsdiff, and rcsmerge overrides this default. Beware rcs -kv, because -kv is incompatible with co -l. Use rcs -kkv to restore the normal default keyword substitution.

-l[rev]
Lock the revision with number rev. If a branch is given, lock the latest revision on that branch. If rev is omitted, lock the latest revision on the default branch. Locking prevents overlapping changes. A lock is removed with ci or rcs -u (see below).

-u[rev]
Unlock the revision with number rev. If a branch is given, unlock the latest revision on that branch. If rev is omitted, remove the latest lock held by the caller. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated by end-of-file or by a line containing . by itself.

-L Set locking to strict. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

-U Set locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should not be used for files that are shared. Whether default locking is strict is

determined by your system administrator, but it is normally strict.

-mrev:msg
Replace revision rev's log message with msg.

-nname[:[rev]]
Associate the symbolic name name with the branch or revision rev. Delete the symbolic name if both : and rev are omitted; otherwise, print an error message if name is already associated with another number. If rev is symbolic, it is expanded before association. A rev consisting of a branch number followed by a . stands for the current latest revision in the branch. A : with an empty rev stands for the current latest revision on the default branch, normally the trunk. For example, rcs -nname: RCS/#? associates name with the current latest revision of all the named RCS files; this contrasts with rcs -nname:$ RCS/#? which associates name with the revision numbers extracted from keyword strings in the corresponding working files.

-Nname[:[rev]]
Act like -n, except override any previous assignment of name.

-orange
deletes ("outdates") the revisions given by range. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form rev1:rev2 means revisions rev1 to rev2 on the same branch, :rev means from the beginning of the branch containing rev up to and including rev, and rev: means from revision rev to the end of the branch containing rev. None of the outdated revisions may have branches or locks.

-q Run quietly; do not print diagnostics.

-I Run interactively, even if the standard input is not a terminal.

-sstate[:rev]
Set the state attribute of the revision rev to state . If rev is a branch number, assume the latest revision on that branch. If rev is omitted, assume the latest revision on the default branch. Any identifier is acceptable for state. A useful set of states is Exp (for experimental), Stab (for stable), and Rel (for released). By default,
ci
sets the state of a revision
to Exp.

```
     -t[file]
          Write descriptive text from the contents of  the  named
          file  into  the  RCS  file, deleting the existing text.
          The file pathname may not begin with  -.   If  file  is
          omitted,  obtain  the  text  from  standard input, ter-
          minated by end-of-file or by  a  line  containing  . by
          itself.   Prompt  for the text if interaction is possi-
          ble; see -I.  With -i,  descriptive  text  is  obtained
          even if -t is not given.

     -t-string
          Write descriptive text from the  string  into  the  RCS
          file, deleting the existing text.

     -Vn  Emulate RCS version n.  See
                co
                 for details.

     -xsuffixes
          Use suffixes to characterize RCS files. See
                ci
                 for de-
          tails.

COMPATIBILITY
     The -brev option generates an RCS file that cannot be parsed
     by RCS version 3 or earlier.

     The -ksubst options (except -kkv) generate an RCS file  that
     cannot be parsed by RCS version 4 or earlier.

     Use rcs -Vn to make an RCS file acceptable to RCS version  n
     by discarding information that would confuse version n.

     RCS version 5.5 and earlier does not support the -x  option,
     and requires a ,v suffix on an RCS pathname.

FILES
     rcs accesses files much as
                ci
                 does, except that it uses the
     effective user for all accesses,  it does not write the wor-
     king file or its directory,  and  it  does not even read the
     working file unless a revision number of $ is specified.

ENVIRONMENT
     RCSINIT
          options prepended to the argument  list,  separated  by
          spaces.  See
                ci
                 for details.

DIAGNOSTICS
     The RCS pathname and the revisions outdated are  written  to
     the  diagnostic output.  The exit status is zero if and only
     if all operations were successful.
```

```
IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.6; Release Date: 1991/09/26.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                ,
                ident
                ,
                rcsdiff
                ,
                rcsintro
                ,

                rcsmerge
                ,
                rlog
                ,
                rcsfile
                     Walter  F.  Tichy,  RCS--A  System  for  Version  Control,
     Software--Practice & Experience 15, 7 (July 1985), 637-654.

BUGS
     The separator for revision ranges in the -o option  used  to
     be - instead of :, but this leads to confusion when symbolic
     names contain -.  For backwards compatibility rcs  -o  still
     supports  the  old  -  separator,  but  it  warns about this
     obsolete use.

     Symbolic names need  not  refer  to  existing  revisions  or
     branches.   For  example, the -o option does not remove sym-
     bolic names for the outdated revisions; you must use  -n  to
     remove the names.
```

1.8 RCSCLEAN

```
                NAME
     rcsclean - clean up working files

SYNOPSIS
     rcsclean [options] [ file ... ]

DESCRIPTION
     rcsclean removes working files that  were  checked  out  and
     never  modified.  For each file given, rcsclean compares the
     working file and a revision in the corresponding  RCS  file.
     If  it  finds  a difference, it does nothing.  Otherwise, it
     first unlocks the revision if the -u option  is  given,  and
     then  removes  the  working  file unless the working file is
     writable and the revision is locked.  It logs its actions by
```

outputting the corresponding rcs -u and rm -f commands on the standard output.

If no file is given, all working files in the current directory are cleaned. Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in
ci
.

The number of the revision to which the working file is compared may be attached to any of the options -n, -q, -r, or -u. If no revision number is specified, then if the -u option is given and the caller has one revision locked, rcsclean uses that revision; otherwise rcsclean uses the latest revision on the default branch, normally the root.

rcsclean is useful for clean targets in Makefiles. See also

rcsdiff
, which prints out the differences, and
ci
, which
normally asks whether to check in a file if it was not changed.

OPTIONS

-ksubst
Use subst style keyword substitution when retrieving the revision for comparison. See
co
for details.

-n[rev]
Do not actually remove any files or unlock any revisions. Using this option will tell you what rcsclean would do without actually doing it.

-q[rev]
Do not log the actions taken on standard output.

-r[rev]
This option has no effect other than specifying the revision for comparison.

-u[rev]
Unlock the revision if it is locked and no difference is found.

-Vn Emulate RCS version n. See
co
for details.

-xsuffixes
Use suffixes to characterize RCS files. See
ci
for de-

```
          tails.

EXAMPLES
          rcsclean  #?.c  #?.h

     removes all working files ending in .c or .h that  were  not
     changed since their checkout.

          rcsclean

     removes all working files in the current directory that were
     not changed since their checkout.

FILES
     rcsclean accesses files much as
                ci
                 does.

ENVIRONMENT
     RCSINIT
          options prepended to the argument  list,  separated  by
          spaces.   A  backslash escapes spaces within an option.
          The RCSINIT options are prepended to the argument lists
          of  most  RCS commands.  Useful RCSINIT options include
          -q, -V, and -x.

DIAGNOSTICS
     The exit status is zero if and only if all  operations  were
     successful.    Missing  working  files  and  RCS  files  are
     silently ignored.

IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 1.8; Release Date: 1991/11/03.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                ,
                ident
                ,
                rcs
                ,
                rcsdiff
                ,
                rcsintro
                ,

                rcsmerge
                ,
                rlog
                ,
                rcsfile
```

Walter F. Tichy, RCS--A System for Version Control, Software--Practice & Experience 15, 7 (July 1985), 637-654.

BUGS

At least one file must be given in older Unix versions that do not provide the needed directory scanning operations.

1.9 RCSDIFF

NAME

rcsdiff - compare RCS revisions

SYNOPSIS

rcsdiff [-ksubst] [-q] [-rrev1 [-rrev2]] [-Vn] [-xsuffixes] [diff options] file ...

DESCRIPTION

rcsdiff runs diff to compare two revisions of each RCS file given.

Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in

ci

.

The option -q suppresses diagnostic output. Zero, one, or two revisions may be specified with -r. The option -ksubst affects keyword substitution when extracting revisions, as described in

co

; for example, -kk -r1.1 -r1.2 ignores differences in keyword values when comparing revisions 1.1 and 1.2. To avoid excess output from locker name substitution, -kkvl is assumed if (1) at most one revision option is given, (2) no -k option is given, (3) -kkv is the default keyword substitution, and (4) the working file's mode would be produced by co -l. See

co

for details about -V and -x. Otherwise, all options of diff that apply to regular files are accepted, with the same meaning as for diff.

If both rev1 and rev2 are omitted, rcsdiff compares the latest revision on the default branch (by default the trunk) with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If rev1 is given, but rev2 is omitted, rcsdiff compares revision rev1 of the RCS file with the contents of the corresponding working file.

If both rev1 and rev2 are given, rcsdiff compares revisions rev1 and rev2 of the RCS file.

```
     Both rev1 and rev2 may be given numerically or symbolically.

EXAMPLE
     The command

             rcsdiff  f.c

     compares the latest revision on the default  branch  of  the
     RCS file to the contents of the working file f.c.

ENVIRONMENT
     RCSINIT
          options prepended to the argument  list,  separated  by
          spaces.  See
                ci
                 for details.

DIAGNOSTICS
     Exit status is 0 for no differences during any comparison, 1
     for some differences, 2 for trouble.

IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.3; Release Date: 1991/04/21.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                , diff,
                ident
                ,
                rcs
                ,
                rcsintro
                ,

                rcsmerge
                ,
                rlog
                     Walter  F.  Tichy,  RCS--A  System  for   Version   Control,
     Software--Practice & Experience 15, 7 (July 1985), 637-654.
```

1.10 RCSFREEZE

```
                NAME
     rcsfreeze - freeze a configuration  of  sources  checked  in
     under RCS

SYNOPSIS
     rcsfreeze [name]
```

DESCRIPTION

rcsfreeze assigns a symbolic revision number to a set of RCS files that form a valid configuration.

The idea is to run rcsfreeze each time a new version is checked in. A unique symbolic name (C_number, where number is increased each time rcsfreeze is run) is then assigned to the most recent revision of each RCS file of the main trunk.

An optional name argument to rcsfreeze gives a symbolic name to the configuration. The unique identifier is still generated and is listed in the log file but it will not appear as part of the symbolic revision name in the actual RCS files.

A log message is requested from the user for future reference.

The shell script works only on all RCS files at one time. All changed files must be checked in already. Run

rcsclean
first and see whether any sources remain in the current directory.

FILES

RCS/.rcsfreeze.ver
version number

RCS/.rcsfreeze.log
log messages, most recent first

AUTHOR

Stephan v. Bechtolsheim

SEE ALSO

co
,
rcs
,
rcsclean
,
rlog
BUGS

rcsfreeze does not check whether any sources are checked out and modified.

Although both source file names and RCS file names are accepted, they are not paired as usual with RCS commands.

Error checking is rudimentary.
rcsfreeze is just an optional example shell script, and should not be taken too seriously. See CVS for a more complete solution.

1.11 RCSMERGE

```
            NAME
     rcsmerge - merge RCS revisions

SYNOPSIS
     rcsmerge [options] file

DESCRIPTION
     rcsmerge incorporates the changes between two  revisions  of
     an RCS file into the corresponding working file.

     Pathnames matching an RCS suffix denote RCS files; all  oth-
     ers  denote working files.  Names are paired as explained in

            ci
            .

     At least one revision must be  specified  with  one  of  the
     options  described below, usually -r.  At most two revisions
     may be specified.  If only one revision  is  specified,  the
     latest  revision on the default branch (normally the highest
     branch on the trunk) is assumed  for  the  second  revision.
     Revisions may be specified numerically or symbolically.

     rcsmerge prints a warning if there are overlaps, and  delim-
     its  the  overlapping regions as  explained  in
            merge
            .  The
     command is useful for incorporating changes into a  checked-
     out revision.

OPTIONS
     -ksubst
          Use subst style keyword substitution.  See
                co
                 for  de-
          tails. For example, -kk -r1.1 -r1.2 ignores differences
          in keyword values when merging the  changes from 1.1 to
          1.2.

     -p[rev]
          Send the result to standard output instead of overwrit-
          ing the working file.

     -q[rev]
          Run quietly; do not print diagnostics.

     -r[rev]
          Merge with respect to revision rev.  Here an empty  rev
          stands  for  the latest revision on the default branch,
          normally the head.

     -Vn  Emulate RCS version n.  See
                co
                 for details.
```

```
     -xsuffixes
          Use suffixes to characterize RCS files. See
                ci
                 for de-
          tails.

EXAMPLES
     Suppose you have  released  revision  2.8  of  f.c.   Assume
     furthermore  that  after you complete an unreleased revision
     3.4, you receive updates to release 2.8 from  someone  else.
     To  combine  the updates to 2.8 and your changes between 2.8
     and 3.4, put the updates to 2.8 into file f.c and execute

         rcsmerge  -p  -r2.8  -r3.4  f.c  >f.merged.c

     Then examine f.merged.c.  Alternatively, if you want to save
     the  updates  to 2.8 in the RCS file, check them in as revi-
     sion 2.8.1.1 and execute co -j:

         ci  -r2.8.1.1  f.c
         co  -r3.4  -j2.8:2.8.1.1  f.c

     As another example, the following command undoes the changes
     between  revision  2.4 and 2.8 in your currently checked out
     revision in f.c.

         rcsmerge  -r2.8  -r2.4  f.c

     Note the order of  the  arguments,  and  that  f.c  will  be
     overwritten.

ENVIRONMENT
     RCSINIT
          options prepended to the argument  list,  separated  by
          spaces.  See
                ci
                 for details.

DIAGNOSTICS
     Exit status is 0 for no overlaps, 1 for some overlaps, 2 for
     trouble.

IDENTIFICATION
     Author: Walter F. Tichy.
     Revision Number: 5.3; Release Date: 1991/08/19.
     Copyright © 1982, 1988, 1989 by Walter F. Tichy.
     Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                ,
                ident
                ,
```

```
            merge
            ,
            rcs
            ,
            rcsdiff
            ,

            rcsintro
            ,
            rlog
            ,
            rcsfile
                 Walter F. Tichy, RCS--A System for Version Control,
     Software--Practice & Experience 15, 7 (July 1985), 637-654.
```

1.12 RLOG

```
            NAME
     rlog - print log messages and other information about RCS
     files

SYNOPSIS
     rlog [ options ] file ...

DESCRIPTION
     rlog prints information about RCS files.

     Pathnames matching an RCS suffix denote RCS files; all oth-
     ers denote working files. Names are paired as explained in

            ci
            .

     rlog prints the following information for each RCS file: RCS
     pathname, working pathname, head (i.e., the number of the
     latest revision on the trunk), default branch, access list,
     locks, symbolic names, suffix, total number of revisions,
     number of revisions selected for printing, and descriptive
     text. This is followed by entries for the selected revi-
     sions in reverse chronological order for each branch. For
     each revision, rlog prints revision number, author,
     date/time, state, number of lines added/deleted (with
     respect to the previous revision), locker of the revision
     (if any), and log message. All times are displayed in Coor-
     dinated Universal Time (UTC). Without options, rlog prints
     complete information. The options below restrict this out-
     put.

     -L  Ignore RCS files that have no locks set. This is con-
         venient in combination with -h, -l, and -R.

     -R  Print only the name of the RCS file. This is convenient
         for translating a working pathname into an RCS pathname.

     -h  Print only the RCS pathname, working pathname, head,
```

default branch, access list, locks, symbolic names, and suffix.

-t Print the same as -h, plus the descriptive text.

-b Print information about the revisions on the default branch, normally the highest branch on the trunk.

-ddates
Print information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of dates. A range of the form d1<d2 or d2>d1 selects the revisions that were deposited between d1 and d2 inclusive. A range of the form <d or d> selects all revisions dated d or earlier. A range of the form d< or >d selects all revisions dated d or later. A range of the form d selects the single, latest revision dated d or earlier. The date/time strings d, d1, and d2 are in the free format explained in
co
. Quoting is normally
necessary, especially for < and >. Note that the separator is a semicolon.

-l[lockers]
Print information about locked revisions only. In addition, if the comma-separated list lockers of login names is given, ignore all locks other than those held by the lockers. For example, rlog -L -R -lwft RCS/#? prints the name of RCS files locked by the user wft.

-r[revisions]
prints information about revisions given in the comma-separated list revisions of revisions and ranges. A range rev1:rev2 means revisions rev1 to rev2 on the same branch, :rev means revisions from the beginning of the branch up to and including rev, and rev: means revisions starting with rev to the end of the branch containing rev. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range. A branch followed by a . means the latest revision in that branch. A bare -r with no revisions means the latest revision on the default branch, normally the trunk.

-sstates
prints information about revisions whose state attributes match one of the states given in the comma-separated list states.

-w[logins]
prints information about revisions checked in by users with login names appearing in the comma-separated list logins. If logins is omitted, the user's login is assumed.

-Vn Emulate RCS version n when generating logs. See

```
                co
                 for
        more.

    -xsuffixes
        Use suffixes to characterize RCS files. See
                ci
                 for det-
        ails.

    rlog prints the intersection of the revisions selected  with
    the  options  -d, -l, -s, and -w, intersected with the union
    of the revisions selected by -b and -r.

EXAMPLES
        rlog  -L  -R  RCS/#?
        rlog  -L  -h  RCS/#?
        rlog  -L  -l  RCS/#?
        rlog  RCS/#?

    The first command prints the names of all RCS files  in  the
    subdirectory RCS that have locks.  The second command prints
    the headers of those files, and the third prints the headers
    plus  the  log  messages  of the locked revisions.  The last
    command prints complete information.

ENVIRONMENT
    RCSINIT
         options prepended to the argument  list,  separated  by
         spaces.  See
                ci
                 for details.

DIAGNOSTICS
    The exit status is zero if and only if all  operations  were
    successful.

IDENTIFICATION
    Author: Walter F. Tichy.
    Revision Number: 5.3; Release Date: 1991/08/22.
    Copyright © 1982, 1988, 1989 by Walter F. Tichy.
    Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

                co
                ,
                ci
                ,
                ident
                ,
                rcs
                ,
                rcsdiff
                ,

                rcsintro
```

```
,
rcsmerge
,
rcsfile
     Walter F. Tichy, RCS--A System for Version Control,
Software--Practice & Experience 15, 7 (July 1985), 637-654.

BUGS
     The separator for revision ranges in the -r option used to
     be - instead of :, but this leads to confusion when symbolic
     names contain -. For backwards compatibility rlog -r still
     supports the old - separator, but it warns about this
     obsolete use.
```

1.13 RCSFILE

```
          NAME
     rcsfile - format of RCS file

DESCRIPTION
     An RCS file's contents are described by the grammar below.

     The text is free format: space, backspace, tab, newline,
     vertical tab, form feed, and carriage return (collectively,
     white space) have no significance except in strings. How-
     ever, an RCS file must end in a newline character.

     Strings are enclosed by @. If a string contains a @, it
     must be doubled; otherwise, strings may contain arbitrary
     binary data.

     The meta syntax uses the following conventions: `|' (bar)
     separates alternatives; `{' and `}' enclose optional
     phrases; `{' and `}*' enclose phrases that may be repeated
     zero or more times; `{' and '}+' enclose phrases that must
     appear at least once and may be repeated; Terminal symbols
     are in boldface; nonterminal symbols are in italics.

     rcstext    ::=  admin {delta}* desc {deltatext}*

     admin      ::=  head       {num};
                     { branch   {num}; }
                     access     {id}*;
                     symbols    {id : num}*;
                     locks      {id : num}*;  {strict  ;}
                     { comment  {string}; }
                     { expand   {string}; }
                     { newphrase }*

     delta      ::=  num
                     date       num;
                     author     id;
                     state      {id};
                     branches   {num}*;
                     next       {num};
```

```
                { newphrase }*

desc        ::= desc        string

deltatext   ::= num
                log         string
                { newphrase }*
                text        string

num         ::= {digit{.}}+

digit       ::= 0 | 1 | ... | 9

id          ::= letter{idchar}*

letter      ::= any letter

idchar      ::= any visible graphic character except special

special     ::= $ | , | . | : | ; | @

string      ::= @{any character, with @ doubled}*@

newphrase   ::= id word* ;

word        ::= id | num | string | :
```

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap. In most environments RCS uses the ISO 8859/1 encoding: letters are octal codes 101-132, 141-172, 300-326, 330-366 and 370-377, visible graphic characters are codes 041-176 and 240-377, and white space characters are codes 010-015 and 040.

The newphrase productions in the grammar are reserved for future extensions to the format of RCS files. No newphrase will begin with any keyword already in use.

The delta nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the trunk, and are linked through the next field in order of decreasing numbers. The head field in the admin node points to the head of that sequence (i.e., contains the highest pair). The branch node in the admin node indicates the default branch (or revision) for most RCS operations. If empty, the default branch is the highest branch on the trunk.

All delta nodes whose numbers consist of 2n fields (n) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first 2n-1 number fields are identical are linked through the next field in order of increasing numbers. For each such sequence, the delta node whose number is identical to the first 2n-2 number fields of the deltas on that sequence is called the branchpoint. The branches field of a node contains a list of the numbers of

the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

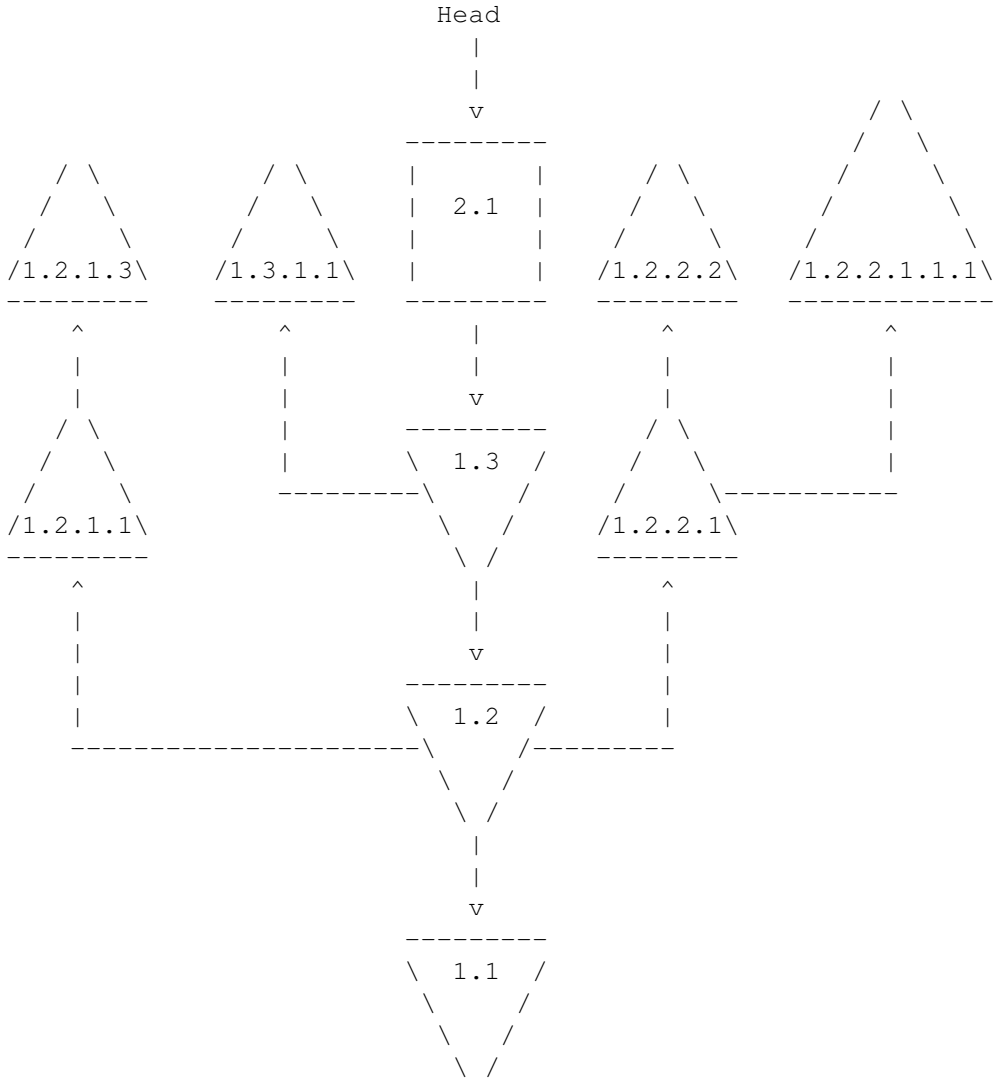

Fig. 1: A revision tree

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 5.1; Release Date: 1991/08/19.

SEE ALSO

co
,
ci
,
ident
,

rcs
,
rcsdiff
,
rcsintro
,

rlog

Walter F. Tichy, RCS--A System for Version Control, Software--Practice & Experience 15, 7 (July 1985), 637-654.