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**(1)). A lock held by someone else can 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**, **–I**, **–j**, **–k**, **–l**, **–M**, **–q**, **–r**, or **–u**. *rev* can be symbolic, numeric, or mixed. Symbolic names in *rev* must already be defined; see the **–n** and **–N** options for assigning names during checkin. If *rev* is **$**, **ci** determines the revision number from keyword values in the working file.

If *rev* begins with a period, then the default branch (normally the trunk) is prepended to it. If *rev* is a branch number followed by a period, then the latest revision on that branch is used.

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**(1)).

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

OPTIONS

–r*rev* Check in revision *rev*.

−r The bare **−r** option (without any revision) has an unusual meaning in **ci**. With other RCS commands, a bare **−r** option specifies the most recent revision on the default branch, but with **ci**, a bare **−r** option reestablishes the default behavior of releasing a lock and removing the working file, and is used to override any default **−l** or **−u** options established by shell aliases or scripts.

−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**, bare **−r**, and **−u** options are mutually exclusive and silently override each other. For example, **ci −u −r** is equivalent to **ci −r** because bare **−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**(1)), 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 can 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*] initial checkin; report an error if the RCS file already exists. This avoids race conditions in certain applications.

−j[*rev*] just checkin and do not initialize; report an error if the RCS file does not already exist.

−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**(1). 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**(1).

−m*msg* uses the string *msg* as the log message for all revisions checked in. By convention, log messages that start with **#** are comments and are ignored by programs like GNU Emacs's **vc** package. Also, log messages that start with **{***clumpname***}** (followed by white space) are meant to be clumped together if possible, even if they are associated with different files; the **{***clumpname***}** label is used only for clumping, and is not considered to be part of the log message itself.

−n*name*

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.

−N*name*

same as **−n**, except that it overrides a previous assignment of *name*.

−s*state* sets the state of the checked-in revision to the identifier *state*. The default state is **Exp**.

−t*file* writes descriptive text from the contents of the named *file* into the RCS file, deleting the existing text. The *file* cannot 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.

−T Set the RCS file's modification time to the new revision's time if the former precedes the latter and there is a new revision; preserve the RCS file's modification time otherwise. If you have locked a revision, **ci** usually updates the RCS file's modification time to the current time, because the lock is stored in the RCS file and removing the lock requires changing the RCS file. This can create an RCS file newer than the working file in one of two ways: first, **ci −M** can create a working file with a date before the current time; second, when reverting to the previous revision the RCS file can change while the working file remains unchanged. These two cases can cause excessive recompilation caused by a **make**(1) dependency of the working file on the RCS file. The **−T** option inhibits this recompilation by lying about the RCS file's date. Use this option with care; it can suppress recompilation even when a checkin of one working file should affect another working file associated with the same RCS file. For example, suppose the RCS file's time is 01:00, the (changed) working file's time is 02:00, some other copy of the working file has a time of 03:00, and the current time is 04:00. Then **ci −d −T** sets the RCS file's time to 02:00 instead of the usual 04:00; this causes **make**(1) to think (incorrectly) that the other copy is newer than the RCS file.

−w*login*

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

−V Print RCS's version number.

−V*n* Emulate RCS version *n*. See **co**(1) for details.

−x*suffixes*

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/***path* or *path1***/RCS/***path2.* 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 filenames, and is empty (i.e. just the empty suffix) for other hosts.

−z*zone* specifies the date output format in keyword substitution, and specifies the default time zone for *date* in the **−d***date* option. The *zone* should be empty, a numeric UTC offset, or the special string **LT** for local time. The default is an empty *zone*, which uses the traditional RCS format of UTC without any time zone indication and with slashes separating the parts of the date; otherwise, times are output in ISO 8601 format with time zone indication. For example, if local time is January 11, 1990, 8pm Pacific Standard Time, eight hours west of UTC, then the time is output as follows:

option	*time output*	
−z	**1990/01/12 04:00:00**	*(default)*
−zLT	**1990-01-11 20:00:00−08**	
−z+05:30	**1990-01-12 09:30:00+05:30**	

The **−z** option does not affect dates stored in RCS files, which are always UTC.

FILE NAMING

Pairs of RCS files and working files can 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 start with **RCS/** or must contain **/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.

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

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

Temporary files are 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.

SETUID USE

To prevent anybody but their RCS administrator from deleting revisions, a set of users can employ setuid privileges as follows.

- Check that the host supports RCS setuid use. Consult a trustworthy expert if there are any doubts. It is best if the **seteuid** system call works as described in Posix 1003.1a Draft 5, because RCS can switch back and forth easily between real and effective users, even if the real user is **root**. If not, the second best is if the **setuid** system call supports saved setuid (the {_POSIX_SAVED_IDS} behavior of Posix 1003.1-1990); this fails only if the real or effective user is **root**. If RCS detects any failure in setuid, it quits immediately.
- Choose a user *A* to serve as RCS administrator for the set of users. Only *A* can invoke the **rcs** command on the users' RCS files. *A* should not be **root** or any other user with special powers. Mutually suspicious sets of users should use different administrators.
- Choose a pathname *B* to be a directory of files to be executed by the users.
- Have *A* set up *B* to contain copies of **ci** and **co** that are setuid to *A* by copying the commands from their standard installation directory *D* as follows:

  ```
  mkdir B
  cp D/c[io] B
  chmod go−w,u+s B/c[io]
  ```

- Have each user prepend *B* to their path as follows:

  ```
  PATH=B:$PATH; export PATH # ordinary shell
  set path=(B $path) # C shell
  ```

- Have *A* create each RCS directory *R* with write access only to *A* as follows:

  ```
  mkdir R
  chmod go−w R
  ```

- If you want to let only certain users read the RCS files, put the users into a group *G*, and have *A* further protect the RCS directory as follows:

  ```
  chgrp G R
  chmod g−w,o−rwx R
  ```

- Have *A* copy old RCS files (if any) into *R*, to ensure that *A* owns them.
- An RCS file's access list limits who can check in and lock revisions. The default access list is empty, which grants checkin access to anyone who can read the RCS file. If you want limit checkin access, have *A* invoke **rcs −a** on the file; see **rcs**(1). In particular, **rcs −e −a***A* limits access to just *A*.
- Have *A* initialize any new RCS files with **rcs −i** before initial checkin, adding the **−a** option if you want to limit checkin access.
- Give setuid privileges only to **ci**, **co**, and **rcsclean**; do not give them to **rcs** or to any other command.
- Do not use other setuid commands to invoke RCS commands; setuid is trickier than you think!

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**, **−x**, and **−z**.

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.
Manual Page Revision: 5.17; Release Date: 1995/06/16.

SEE ALSO

co(1), emacs(1), ident(1), make(1), rcs(1), rcsclean(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), setuid(2), rcsfile(5)

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