NAME

cio - a pair of RCS user interface programs

SYNOPSIS

cii [cii options] [ci options] [filename ...] [dirname ...]

coo [coo options] [co options] [filename ...] [dirname ...]

DESCRIPTION

Cii and *coo* are two programs that will provide an interface to the RCS programs *ci* and *co.* They add:

Recursively check in/out directory structures

Store RCS files in a separate directory structure

Default stores only ASCII files, enabling program to be run on directories with both source and binary

Command line options include:

-A All files. Forces a "cp" of files that are non-ASCII.

-B Handle binary files. Non-ASCII files will initially be checked in by invoking 'rcs -i -kb ...'

-H Insert RCS header. If a valid RCS header not found, a template header will be inserted at the beginning of the file being checked in.

-N No operation. Causes *cii* or *coo* to display the action that would have resulted. Nothing is executed.

-R Recursively walk down directories to check in/out

-T Force title request at start of program (Default only gets title when required.)

-U Update source directory. During a check-in, a copy is made to the directory structure specified by $RCSSRC. If $RCSSRC is not defined, $HOME will be used instead.

-V Verbose. Be real talkative about the work being done.

[all ci/co options]
Passes all other options on to *ci* or *co.*

filename
Optional file names. If not provided, the *cii* program will take all files in the current directory, or *coo* will take all files saved in the RCS directory. (Binary files only if -A was also specified.)

dirname
Both programs will take a directory name as an argument, however: the -R option must also be specified for it to work.

ENVIRONMENT

RCSDIR
If defined, names a path that will be used to store the RCS files. Default is $HOME/RCS. Final path is computed by removing $HOME and/or $RCSWORK from current path.

RCSWORK
If defined, an alternate prefix for current working directory. Allows having multiple directory structures with different prefix's.

HOME
Must be defined as the users home directory.

RCSSRC

If defined and -U flag is specified, during a *cii* procedure, a copy of what's being checked in will be made in the directory structure starting at $RCSSRC. If not specified, *cii* uses $HOME instead.

RCSHEAD

If -H flag is specified, *cii* finds template header from $RCSHEAD directory.

CIOBINEXT

This is a list of the extensions of binary files, including the points and without spaces. It is used to decide wether a file is a binary one or not in case there is no file command (i.e. OS/2).

SAMPLES

A user with a home directory of /usr/bog, has a directory structure called sample/arix. He want's to store all of the files in /usr/bog/sample/arix into the RCS system. If neither RCSDIR or RCSWORK is defined, the RCS path defaults to his home directory, followed by RCS, followed by the current path. So:

```
Current directory   :  /usr/bog/sample/arix
Home directory      :  /usr/bog
RCSDIR              :
RCSWORK             :
RCSSRC              :

Final RCS storage   :  /usr/bog/RCS/sample/arix
Final source storage:  /usr/bog/sample/arix
```

The net effect of this command is to create the RCS directory, then to duplicate the directory structure in a defined place. In this case, since the RCSDIR was not defined, it defaulted to the users home directory.

It is not necessary to build the directories in RCS, the program will build all necessary directories (Including the RCS dir, if needed).

Another example: All RCS files should reside in /usr/RCS, so the above example turns into:

```
Current directory   :  /usr/bog/sample/arix
Home directory      :  /usr/bog
RCSDIR              :  /usr/RCS
RCSWORK             :
RCSSRC              :

Final RCS storage   :  /usr/RCS/sample/arix
Final source storage:  /usr/bog/sample/arix
```

Here is an example showing use of the RCSWORK variable. Some systems may have more than one person working on file. In this case, the path names will have to be similar, but only to a point. Example, /usr/tog is also working in a directory, called sample/arix. His system would look like:

```
Current directory   :  /usr/tog/sample/arix
Home directory      :  /usr/tog
RCSDIR              :  /usr/RCS
RCSWORK             :
RCSSRC              :

Final RCS storage   :  /usr/RCS/sample/arix
Final source storage:  /usr/tog/sample/arix
```

Note that the RCS dir is the same for him. Now, let's take a user who has decided to work somewhere other than his home directory.

```
Current directory   :  /usr/src/rog/sample/arix
Home directory      :  /usr/rog
RCSDIR              :  /usr/RCS
RCSWORK             :  /usr/src/rog
RCSSRC              :

Final RCS storage   :  /usr/RCS/sample/arix
Final source storage:  /usr/rog/sample/arix
```

If RCSSRC is specified to keep the current source (very useful to when one wants to browse through the current source files) in a directory other than his HOME.

```
Current directory   :  /usr/src/rog/sample/arix
Home directory      :  /usr/rog
RCSDIR              :  /usr/RCS
RCSWORK             :  /usr/src/rog
RCSSRC              :  /usr/group

Final RCS storage   :  /usr/RCS/sample/arix
Final source storage:  /usr/group/sample/arix
```

The RCSWORK variable was removed from the current path, before the directory structure was defined. Thus, it is possible to be working in just about anywhere on the system and still use the same directory structure and RCS files.

If you want to recover a directory (or multiple ones) into a new working directory, simply create whatever part of the path you need, set the RCSWORK variable to be the first part of the path, and type "coo [-R]".

Example: A new user (/usr/log) has decided to examine the sample/* files. Here are the steps:

```
Current working directory: /usr/llog
mkdir sample
chdir sample
RCSDIR=/usr/RCS  export RCSDIR
RCSWORK=/usr/log  export RCSWORK
```

This is the final setup:

```
Current directory:  /usr/log/sample
Home directory   :  /usr/llog
RCSDIR           :  /usr/RCS
RCSWORK          :  /usr/log

Final RCS storage:  /usr/RCS/sample/arix
```

SECURITY

Secure archives

Cii/coo will also secure archives. Changing ownership of the cii or coo program to root, and making it suid will allow private archives. With this option, an additional environment variable is searched for: "RCSOWN". If this is not found, the default name "rcsfiles" is used in the following step.

The user name is searched for in the /etc/password. If not found, the real users UID will be used instead. In this fashion, the ci and co programs will be called with the appropriate uses abilities to create directories, and save files. Only the user who owns the RCS files has write ability without going through the cii or coo programs.

The only condition that allows the program to remain in the root owned mode is to have the rcsfiles have a root account. Otherwise, it moves out of the root as soon as it's found the user.

OTHER NOTES

Cii will not unlink files not owned by the user who is checking in the files. This prevents users from deleting files not owned by them, possibly causing harm. (I.E., checking in the /etc/passwd file.) *coo* does not run as root, it runs as the real user. Hence, it is not possible for a user to overwrite files or directories they normally do not have write access to.

Cii/coo only will allow extraction of code by the users in the same group as the user who checked the sources in.

We highly suggest that you exmine the code in the security area if you plan on running it secure. We've tried it secure, and it seems to work. However... We are by no means the "great U*IX hackers", so there is probably some way to run the program that we missed that can allow others access to restricted files. Check it out! If you *do* find something we missed, please send us mail.

EDITOR COMMANDS

The input editor for the logfile entries and title file entries have tilde (˜) command options available. These are:

```
? - Print a help menu.
. - End of input.
! - Invoke a user shell. Note: Shell is invoked
    as the real user-id; No arguments are passed
    or allowed. The ENV variable "SHELL" is
    searched for, if not present /bin/sh will be
    invoked.
e - Edit the message using a default editor.
    Searches for the ENV variable VISUAL or
    EDITOR. If not found, /usr/bin/vi is the
    default editor.
p - Print message buffer. Displays the current
    contents of the message buffer.
r - Read in a file. Requires file name as an
    argument. Named file will be appended to
    current buffer.
w - Write message to a file. Requires file name
    as an argument. Appends message buffer to
    file name given. If file does not exist,
    creates file. Does not create directories.
```

AUTHORS

Jason P. Winters (jason@grinch.uucp) and Craig J. Kim (cjkim@aeras.uucp)

FILES

The following template files can be created to insert RCS header into source files during *cii* process with -H option. The location for these files is specified by setting $RCSHEAD environment variable. If $RCSHEAD is not set, $HOME will be used instead. File types are determined by examining the content via file(1) program and by file extensions (e.g. .c for C programs, .mk for makefiles, .1 for nroff, and .s for assembly).

```
.rcshead     - default template
.rcshead.c   - C program template
```

.rcshead.s - Assembly source template
.rcshead.sh - Shell script template
.rcshead.f - Fortran program template
.rcshead.mk - makefile template
.rcshead.h - C header file template
.rcshead.roff - nroff, troff, man file template

DIAGNOSTICS

Linking cio to ciitest or cootest will cause the program to parse the directory paths, and print out the final pathnames. Nothing else happens. Using the -N option will cause the program to execute as normal, except that no files or directories will be created.

BUGS

On some systems, the Control-D as end of input to a log or title entry can cause problems with STDIN. We've added a call to clearerr(), but have not tested if that fixes it. (It doesn't break it, so...)

Any other bugs we would like to hear about. We might even make a new release, if people actually use this. :)

AUTHOR

Jason Winters, grinch!jason