Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'cp.1p' command

$ man cp.1p

CP(1P) POSIX Programmer's Manual CP(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cp ? copy files

SYNOPSIS

cp [-Pfip] source_file target_file

cp [-Pfip] source_file... target

cp -R [-H|-L|-P] [-fip] source_file... target

DESCRIPTION

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The cp utility shall copy the contents of source_file (or, if source_file is a file of type symbolic link, the contents of the file referenced by source_file) to the desti? nation path named by target_file.

The second synopsis form is denoted by two or more operands where the -R option is not specified and the first synopsis form is not applica? ble. It shall be an error if any source_file is a file of type direc? tory, if target does not exist, or if target does not name a directory.

The cp utility shall copy the contents of each source_file (or, if

source_file is a file of type symbolic link, the contents of the file referenced by source_file) to the destination path named by the con? catenation of target, a single <slash> character if target did not end in a <slash>, and the last component of source_file.

The third synopsis form is denoted by two or more operands where the -R option is specified. The cp utility shall copy each file in the file hierarchy rooted in each source_file to a destination path named as follows:

* If target exists and names an existing directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of target, a single <slash> character if target did not end in a <slash>, and the pathname of the file rela? tive to the directory containing source_file.
* If target does not exist and two operands are specified, the name of the corresponding destination path for source_file shall be tar? get; the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of target, a <slash> character, and the pathname of the file relative to source_file.

It shall be an error if target does not exist and more than two oper? ands are specified, or if target exists and does not name a directory.

In the following description, the term dest_file refers to the file named by the destination path. The term source_file refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a source_file operand. If source_file is a file of type symbolic link:

* If the -R option was not specified, cp shall take actions based on the type and contents of the file referenced by the symbolic link, and not by the symbolic link itself, unless the -P option was spec? ified.
* If the -R option was specified:
  -- If none of the options -H, -L, nor -P were specified, it is un? specified which of -H, -L, or -P will be used as a default.

-- If the -H option was specified, cp shall take actions based on the type and contents of the file referenced by any symbolic link specified as a source_file operand.

-- If the -L option was specified, cp shall take actions based on the type and contents of the file referenced by any symbolic link specified as a source_file operand or any symbolic links encountered during traversal of a file hierarchy.

-- If the -P option was specified, cp shall copy any symbolic link specified as a source_file operand and any symbolic links en? countered during traversal of a file hierarchy, and shall not follow any symbolic links.

For each source_file, the following steps shall be taken:

1. If source_file references the same file as dest_file, cp may write a diagnostic message to standard error; it shall do nothing more with source_file and shall go on to any remaining files.

2. If source_file is of type directory, the following steps shall be taken:

   a. If the -R option was not specified, cp shall write a diagnostic message to standard error, do nothing more with source_file, and go on to any remaining files.

   b. If source_file was not specified as an operand and source_file is dot or dot-dot, cp shall do nothing more with source_file and go on to any remaining files.

   c. If dest_file exists and it is a file type not specified by the System Interfaces volume of POSIX.1?2017, the behavior is im? plementation-defined.

   d. If dest_file exists and it is not of type directory, cp shall write a diagnostic message to standard error, do nothing more with source_file or any files below source_file in the file hi? erarchy, and go on to any remaining files.

   e. If the directory dest_file does not exist, it shall be created with file permission bits set to the same value as those of source_file, modified by the file creation mask of the user if

the -p option was not specified, and then bitwise-inclusively OR'ed with S_IRWXU. If dest_file cannot be created, cp shall write a diagnostic message to standard error, do nothing more with source_file, and go on to any remaining files. It is un? specified if cp attempts to copy files in the file hierarchy rooted in source_file.

f. The files in the directory source_file shall be copied to the directory dest_file, taking the four steps (1 to 4) listed here with the files as source_files.

g. If dest_file was created, its file permission bits shall be changed (if necessary) to be the same as those of source_file, modified by the file creation mask of the user if the -p option was not specified.

h. The cp utility shall do nothing more with source_file and go on to any remaining files.

3. If source_file is of type regular file, the following steps shall be taken:

a. The behavior is unspecified if dest_file exists and was written by a previous step. Otherwise, if dest_file exists, the follow? ing steps shall be taken:

i. If the -i option is in effect, the cp utility shall write a prompt to the standard error and read a line from the standard input. If the response is not affirmative, cp shall do nothing more with source_file and go on to any remaining files.

ii. A file descriptor for dest_file shall be obtained by per? forming actions equivalent to the open() function defined in the System Interfaces volume of POSIX.1?2017 called us? ing dest_file as the path argument, and the bitwise-inclu? sive OR of O_WRONLY and O_TRUNC as the oflag argument.

iii. If the attempt to obtain a file descriptor fails and the -f option is in effect, cp shall attempt to remove the file by performing actions equivalent to the unlink()

function defined in the System Interfaces volume of POSIX.1?2017 called using dest_file as the path argument.

If this attempt succeeds, cp shall continue with step 3b.

b. If dest_file does not exist, a file descriptor shall be ob? tained by performing actions equivalent to the open() function defined in the System Interfaces volume of POSIX.1?2017 called using dest_file as the path argument, and the bitwise-inclusive OR of O_WRONLY and O_CREAT as the oflag argument. The file per? mission bits of source_file shall be the mode argument.

c. If the attempt to obtain a file descriptor fails, cp shall write a diagnostic message to standard error, do nothing more with source_file, and go on to any remaining files.

d. The contents of source_file shall be written to the file de? scriptor. Any write errors shall cause cp to write a diagnostic message to standard error and continue to step 3e.

e. The file descriptor shall be closed.

f. The cp utility shall do nothing more with source_file. If a write error occurred in step 3d, it is unspecified if cp con? tinues with any remaining files. If no write error occurred in step 3d, cp shall go on to any remaining files.

4. Otherwise, the -R option was specified, and the following steps shall be taken:

a. The dest_file shall be created with the same file type as source_file.

b. If source_file is a file of type FIFO, the file permission bits shall be the same as those of source_file, modified by the file creation mask of the user if the -p option was not specified.

Otherwise, the permissions, owner ID, and group ID of dest_file are implementation-defined.

If this creation fails for any reason, cp shall write a diag? nostic message to standard error, do nothing more with source_file, and go on to any remaining files.

c. If source_file is a file of type symbolic link, and the options

require the symbolic link itself to be acted upon, the pathname contained in dest_file shall be the same as the pathname con? tained in source_file.

If this fails for any reason, cp shall write a diagnostic mes? sage to standard error, do nothing more with source_file, and go on to any remaining files.

If the implementation provides additional or alternate access control mechanisms (see the Base Definitions volume of POSIX.1?2017, Section 4.5, File Access Permissions), their effect on copies of files is im? plementation-defined.

OPTIONS

The cp utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-f If a file descriptor for a destination file cannot be ob? tained, as described in step 3.a.ii., attempt to unlink the destination file and proceed.

-H Take actions based on the type and contents of the file ref? erenced by any symbolic link specified as a source_file oper? and.

-i Write a prompt to standard error before copying to any exist? ing non-directory destination file. If the response from the standard input is affirmative, the copy shall be attempted; otherwise, it shall not.

-L Take actions based on the type and contents of the file ref? erenced by any symbolic link specified as a source_file oper? and or any symbolic links encountered during traversal of a file hierarchy.

-P Take actions on any symbolic link specified as a source_file operand or any symbolic link encountered during traversal of a file hierarchy.

-p Duplicate the following characteristics of each source file in the corresponding destination file:

1. The time of last data modification and time of last ac? cess. If this duplication fails for any reason, cp shall write a diagnostic message to standard error.
2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether cp writes a diag? nostic message to standard error.
3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be dupli? cated as well. If this duplication fails for any reason, cp shall write a diagnostic message to standard error.

If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not dupli? cated in the destination file, it is unspecified whether cp writes a diagnostic message to standard error.

The order in which the preceding characteristics are dupli? cated is unspecified. The dest_file shall not be deleted if these characteristics cannot be preserved.

-R Copy file hierarchies.

Specifying more than one of the mutually-exclusive options -H, -L, and -P shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operands shall be supported:

source_file

A pathname of a file to be copied. If a source_file operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard input.

target_file

A pathname of an existing or nonexistent file, used for the output when a single file is copied. If a target_file operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard output.

target A pathname of a directory to contain the copied files.

STDIN

The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.

INPUT FILES

The input files specified as operands may be of any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of cp:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the yesexpr locale keyword in the LC_MESSAGES category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the ex? tended regular expression defined for the yesexpr locale key? word in the LC_MESSAGES category.

LC_MESSAGES

Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of di? agnostic messages and prompts written to standard error.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

A prompt shall be written to standard error under the conditions speci? fied in the DESCRIPTION section. The prompt shall contain the destina? tion pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files may be of any type.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All files were copied successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If cp is prematurely terminated by a signal or error, files or file hi? erarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.

The following sections are informative.

APPLICATION USAGE

The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID or set-group-ID to them when copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users has execute permission to a set-user-ID program than did previously. In particular, this is a problem for superusers copying users' trees.

EXAMPLES

None.

RATIONALE

The -i option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3 BSD version does not prompt if the standard input is not a terminal, the standard developers decided that use of -i is a request for inter? action, so when the destination path exists, the utility takes instruc? tions from whatever responds on standard input.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because imple? mentations may desire more descriptive prompts than those used on his? torical implementations. Therefore, an application using the -i option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The -p option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. This volume of POSIX.1?2017 extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This vol? ume of POSIX.1?2017 requires that the modification times be preserved.

The statement that the order in which the characteristics are dupli? cated is unspecified is to permit implementations to provide the maxi? mum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether cp writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users us? ing -p to duplicate some portion of the file characteristics, indiffer? ent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

Earlier versions of this standard included support for the -r option to copy file hierarchies. The -r option is historical practice on BSD and BSD-derived systems. This option is no longer specified by POSIX.1?2008

but may be present in some implementations. The -R option was added as a close synonym to the -r option, selected for consistency with all other options in this volume of POSIX.1?2017 that do recursive direc? tory descent.

The difference between -R and the removed -r option is in the treatment by cp of file types other than regular and directory. It was implemen? tation-defined how the - option treated special files to allow both historical implementations and those that chose to support -r with the same abilities as -R defined by this volume of POSIX.1?2017. The origi? nal -r flag, for historic reasons, did not handle special files any differently from regular files, but always read the file and copied its contents. This had obvious problems in the presence of special file types; for example, character devices, FIFOs, and sockets.

When a failure occurs during the copying of a file hierarchy, cp is re? quired to attempt to copy files that are on the same level in the hier? archy or above the file where the failure occurred. It is unspecified if cp shall attempt to copy files below the file where the failure oc? curred (which cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation-defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the historical mknod, ln, or other utility to create the file. It is also probable that additional privileges are required to create block, character, or other implementation-defined special file types.

Additionally, the -p option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of cp use the mode of the source file as modified by the file mode creation mask. Other

choices would have been to use the mode of the source file unmodified by the creation mask or to use the same mode as would be given to a new file created by the user (plus the execution bits of the source file) and then modify it by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part re? tained.

When creating directories, historical versions of cp use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that cp can copy trees where the user has read permission, but the owner does not. A side-effect is that if the file creation mask denies the owner per? missions, cp fails. Also, once the copy is done, historical versions of cp set the permissions on the created directory to be the same as the source directory, unmodified by the file creation mask.

This behavior has been modified so that cp is always able to create the contents of the directory, regardless of the file creation mask. After the copy is done, the permissions are set to be the same as the source directory, as modified by the file creation mask. This latter change from historical behavior is to prevent users from accidentally creating directories with permissions beyond those they would normally set and for consistency with the behavior of cp in creating files.

It is not a requirement that cp detect attempts to copy a file to it? self; however, implementations are strongly encouraged to do so. His? torical implementations have detected the attempt in most cases.

There are two methods of copying subtrees in this volume of POSIX.1?2017. The other method is described as part of the pax utility (see pax). Both methods are historical practice. The cp utility pro? vides a simpler, more intuitive interface, while pax offers a finer granularity of control. Each provides additional functionality to the other; in particular, pax maintains the hard-link structure of the hi? erarchy, while cp does not. It is the intention of the standard devel? opers that the results be similar (using appropriate option combina? tions in both utilities). The results are not required to be identical;

there seemed insufficient gain to applications to balance the diffi?
culty of implementations having to guarantee that the results would be
exactly identical.

The wording allowing cp to copy a directory to implementation-defined
file types not specified by the System Interfaces volume of
POSIX.1?2017 is provided so that implementations supporting symbolic
links are not required to prohibit copying directories to symbolic
links. Other extensions to the System Interfaces volume of POSIX.1?2017
file types may need to use this loophole as well.

FUTURE DIRECTIONS

None.

SEE ALSO

mv, find, ln, pax

The Base Definitions volume of POSIX.1?2017, Section 4.5, File Access
Permissions, Chapter 8, Environment Variables, Section 12.2, Utility
Syntax Guidelines

The System Interfaces volume of POSIX.1?2017, open(), unlink()

COPYRIGHT